DEVELOPMENT AND VALIDATION OF REENTRY SIMULATION USING MATLAB

THESIS

Robert E Jameson Jr, Captain, USAF

AFIT/GSS/ENY/06-M08

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GSS/ENY/06-M08

DEVELOPMENT AND VALIDATION OF REENTRY SIMULATION USING MATLAB

THESIS

Presented to the Faculty

Department of Aeronautics and Astronautics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science (Space Systems)

Robert E Jameson Jr, BAAS

Captain, USAF

March 2006

AFIT/GSS/ENY/06-M08

DEVELOPMENT AND VALIDATION OF REENTRY SIMULATION USING MATLAB

Robert E Jameson Jr, BAAS
Captain, USAF

Approved:

_____	_____
Lt Col Kerry Hicks (Chairman)	date
_____	_____
Dr. Richard Cobb (Member)	date
_____	_____
Lt Col Nathan Titus (Member)	date

Abstract

This research effort develops a program using MATLAB to solve the equations of motion for atmospheric reentry and analyzes the validity of the program for use as a tool to expeditiously predict reentry profiles. The reentry vehicle is modeled as a point mass with constant aerodynamic properties as defined by the user. The equations of motion for reentry are based on the two-body problem. The atmosphere is modeled as a single layer exponentially decreasing in density. The MATLAB program has the ability to derive the initial trajectory conditions from the position and velocity relative to the rotating surface of the Earth, the Earth-centered inertial position and velocity, or the classic orbital elements. The program is compared to previously established programs in order to validate its accuracy and numerical stability when predicting various reentry profiles to include sub-orbital, super-circular and hyperbolic trajectories as well as wide ranges of aerodynamic properties.

.

Acknowledgements

I would like to express appreciation to my faculty advisor, Lt Col Kerry Hicks, for his guidance and support throughout the course of this research effort. His insight and excellent instruction on spaceflight and reentry dynamics were invaluable. I would also like to thank my wife, for her unvarying support throughout my career and for her supreme example of fortitude, honesty and grace.

Robert E. Jameson Jr

Table of Contents

Page

List of Figures

Page

Page

List of Tables

Table Page

DEVELOPMENT AND VALIDATION OF REENTRY SIMULATION USING MATLAB

I. Introduction

Background

Over the years several programs have been developed to predict the dynamics of vehicles during atmospheric reentry. The industry and military standard is the Program to Optimize Simulated Trajectories (POST). POST is a generalized point mass, discrete parameter targeting and optimization program. The program was originally developed for NASA in 1970 as a Space Shuttle trajectory optimization program. Since that time it has been repeatedly updated to improve its capabilities in the areas of vehicle modeling, trajectory simulation and targeting optimization. It features modular program construction and generalized routines so that various vehicle and planetary parameters can be simulated. It is based on FORTRAN 77 and C programming languages (Brauer and others, 1977:1-3).

POST provides the capability to optimize point mass trajectories for both powered and unpowered vehicles throughout an entire regime of ascent to orbit and maneuvers to reentering the atmosphere of a rotating, oblate or spherical planet. The program integrates the equations of motion for a user-defined model while solving a constrained optimization problem to maximize or minimize a user defined function. The user supplies a guess for the optimal control vector and POST iteratively solves the problem for the optimal control vector that maximizes a scalar objective function subject to the user defined constraints (Brauer and others, 1977:7,8).

Typical applications of POST include a variety of simulations such as ascent to orbit with optimization of payload, velocity or initial weight; ascent abort with maximization of abort interval; intercontinental ballistic missiles with maximization of payload; reentry with optimization of heat rate or crossrange distance; orbital maneuvers to minimize fuel consumption; and aircraft performance to maximize velocity, cruise time or payload (Brauer and others, 1977:13). Among other projects, POST has recently been used in the planning for the Mars Airplane (Murray, 2001:3), the aerocapture simulation for the Titan Explorer Mission to the Saturnian system (Way, 2003:1) and NASA's Reusable Launch Vehicle.

A program called TARGET was developed as a simpler and faster alternative to POST. TARGET was created for use as a tool in the systems concept stage where speed and reasonable accuracy are sufficient for planning. Although POST can model much more complicated motion than TARGET, trajectories simulated by TARGET have been validated as virtually identical to the results POST generates with the same input parameters. TARGET also generates a trajectory in $1/10^{th}$ the time required by POST. As with POST, TARGET is programmed in FORTRAN. TARGET does show some numerical instability when unrealistic final velocities are targeted and when extreme dynamics such as hyperbolic velocity combined with negative lift are modeled (Hicks,1993:4-1).

TARGET solves the equations of motion for reentry as a two-point boundary-value problem. It uses a "double-loop" shooting algorithm to optimize the simulated trajectory. The "double-loop" method attempts to match the unspecified "free" conditions to the boundary conditions identified by the user by starting with an initial

guess and iteratively solving the two-point boundary-value problem until an optimal solution is found and the "free" conditions are identified. Due to the fact that it is a targeting program, the user must identify the final radius, longitude, latitude, and velocity for the reentry vehicle. Additionally, the user must supply the initial radius, flight-path angle relative to the plane perpendicular to the radius, and the orbital inclination. The program is able to generate a reentry profile to include variations in velocity, altitude, flight-path angle, latitude, longitude, heading, Mach number, and rate of heating (Hicks, 1993:A-1).

A third program of similar nature to the objective of this research is called IMPULSE. The National Air and Space Intelligence Agency (NASIC) developed IMPULSE as a MATLAB based means to simulate ballistic missile flight paths. IMPULSE can model the entire trajectory of missiles from boost through impact. The unclassified version of the program contains examples of missiles and RVs that are not classified and is preloaded with the geographical data of multiple launch and impact sites across the globe. Multiple reentry vehicles (RVs) can be simulated. It also includes the ability to model events during the missile flight that change the mass or aerodynamics of the RV, such as booster separation, shroud ejection, or change in angle of attack (IMPULSE 2005).

The focus of this research is the development and validation of a reentry simulator using the same language (MATLAB) as IMPULSE. Whereas IMPULSE is a complex application that considers the entire trajectory of a ballistic missile with only a rough analysis of the reentry portion, this research will focus on a quick solution for the

simulation of a wide variety of entry trajectories with much smaller integration steps than IMPULSE employs.

MATLAB (MATrix LABoratory) is a high-level technical computing language for algorithm development, data visualization, and data analysis intended for use with numerical computations. The software has a wide range of applications, including signal and image processing, communications, control design, test and measurement, financial modeling and analysis, and computational biology. The original goal was to enable scientists and engineers to use matrix-based techniques to solve problems, without having to write programs in traditional languages like C and FORTRAN. Over the years additional capabilities have been added, in particular outstanding graphics support. MATLAB is rapidly displacing other languages as the software of choice for algorithm programming in education and industry and is now the preferred tool for many agencies of the U. S. Air Force and the Department of Defense (General 2006).

This paper describes the development and validation of the trajectory simulation program. Chapter II begins with a description of the reference frames used to describe the position and direction of travel of the reentry vehicle, followed by development of the equations of motion. Chapter III shows how the initial conditions were established for the equations of motion. Chapter IV presents a comparison of the MATLAB program developed in this research with established software programs and analytical solutions. A description of the program operation is also included. The final chapter (V) states the conclusions drawn from the research and provides recommendations for improvements to the software that may be undertaken in the future. Finally, the Appendices include a flowchart of the alogorithm, descriptions of the MATLAB code and the code itself.

II. Equations of Motion for Reentry

Chapter Overview

The purpose of this chapter is to explain the development of the equations of motion for atmospheric reentry. First, the reference frames used in this research will be established. Next, the equations of motion will be derived for a point mass flying into the atmosphere of a rotating planet. All of the equations and figures in this chapter are summarized from Hicks (2006:33-58).

Reference Frames

The geocentric-equatorial system is an inertial reference frame with its origin at the center of the planet. In this system the x-y plane is the equatorial plane with the x-axis pointing at the vernal equinox. The z-axis passes through the north pole of the planet. The y-axis is positioned such that $\hat{e}_z = \hat{e}_x \times \hat{e}_y$. In the case of the Earth this is commonly referred to as the Earth-Centered Inertial (ECI) reference frame and will be referred to as such in this research.

In the planet-fixed coordinate frame the system spins about the z-axis with the rotation of the planet. In the case of the earth this reference frame will be referred to as the Earth-Centered Fixed (ECF) coordinate system. In the ECF system the x-axis points through the Greenwich Meridian (0^o longitude). The relation between the ECI and ECF coordinate systems is illustrated in Figure (1) where the ECF system is shown as $OX_1Y_1Z_1$, the planet rotates with a constant angular velocity $\vec{\omega}$ and Δt is the sidereal

time. Sidereal time is defined as the time elapsed since the two systems were last aligned.

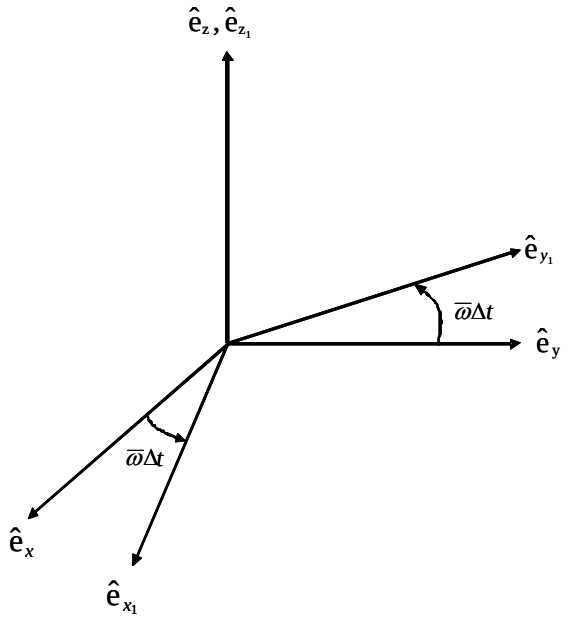

Figure 1. ECF and ECI Coordinate Frames

A coordinate transformation from the ECI frame to the ECF frame is given by:

$$[\hat{\mathbf{e}}_1] = \begin{bmatrix} \cos(\omega\Delta t) & \sin(\omega\Delta t) & 0 \\ -\sin(\omega\Delta t) & \cos(\omega\Delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} [\hat{\mathbf{e}}] \tag{1}$$

A third system also originates at the planet's center but defines the x-axis as pointing along the position vector ($\vec{r}$) of the vehicle of interest. As shown in Figure (2), this frame ($OX_2Y_2X_2$) can be created by a rotation θ about the ECF z-axis followed by a

rotation $-\phi$ about the y-axis. Therefore, θ represents the longitude and ϕ represents the geocentric latitude.

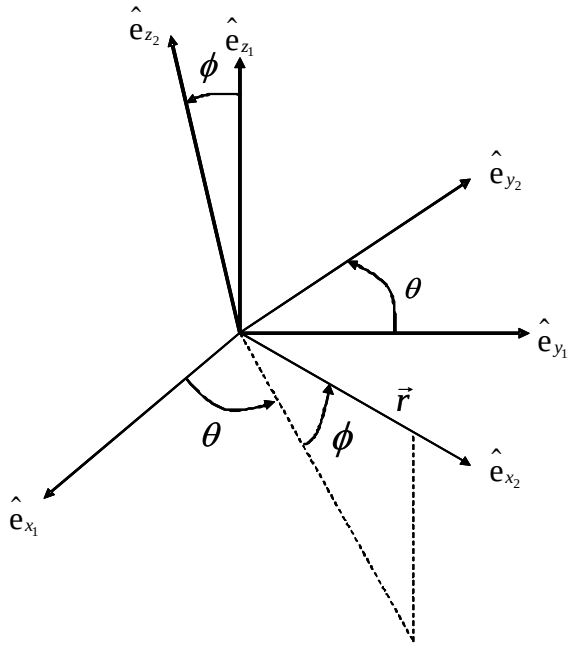

Figure 2. ECF and Vehicle-Pointing Coordinate Frames

Performing the two rotations yields the following relationship between the ECF frame ($OX_1Y_1Z_1$) and the vehicle-pointing frame ($OX_2Y_2Z_2$):

$$\left[\hat{\mathbf{e}}_2\right] = \begin{bmatrix} \cos\phi\cos\theta & \cos\phi\sin\theta & \sin\phi \\ -\sin\theta & \cos\theta & 0 \\ -\sin\phi\cos\theta & -\sin\phi\sin\theta & \cos\phi \end{bmatrix} \left[\hat{\mathbf{e}}_1\right] \tag{2}$$

Figure (3) introduces two more angles necessary to help describe the trajectory for reentry. The flight-path angle, γ, is defined as the angle between the velocity vector, $\vec{V}$, and the plane orthogonal to $\vec{r}$ (the local horizontal plane). γ is positive when $\vec{V}$ is above

the local horizontal plane. The heading angle, ψ, is defined as the angle between the local parallel of latitude and the projection of $\vec{V}$ on the local horizontal plane. ψ is positive in the right-handed direction about the x_2-axis.

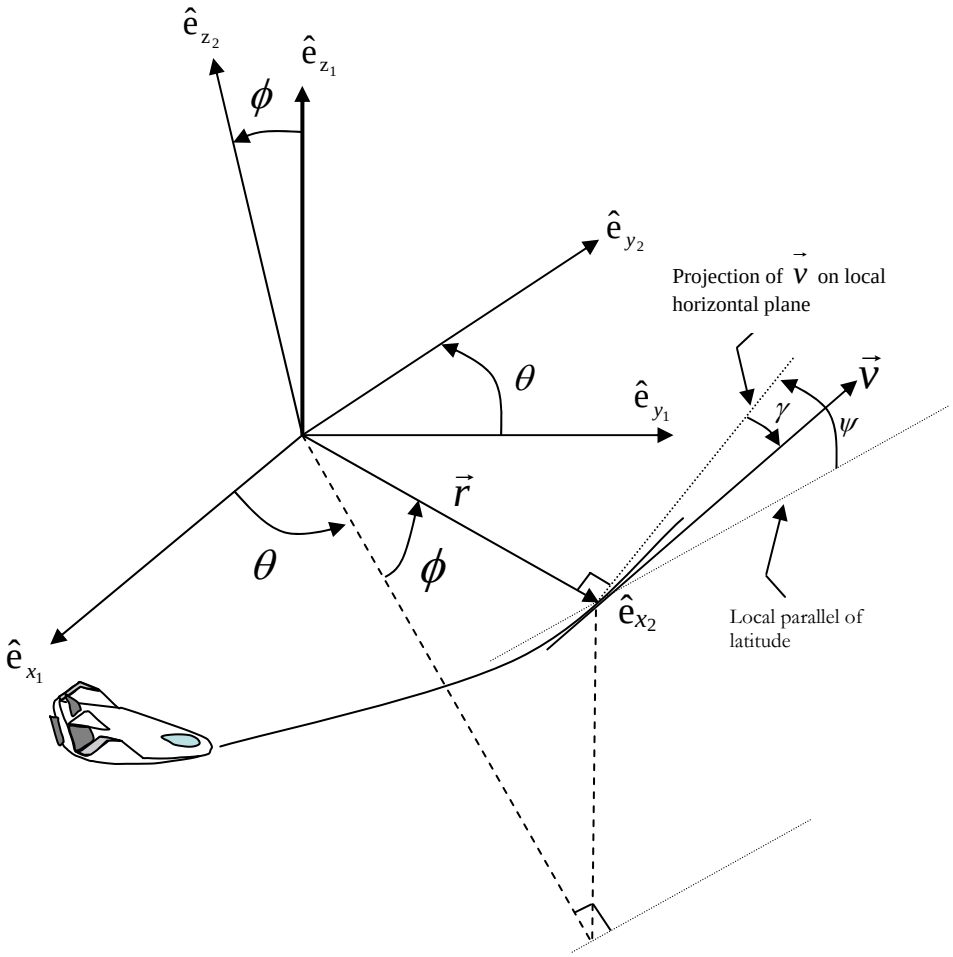

Figure 3. Flight path and heading angles in relation to ECF and Vehicle-Pointing Systems

A useful relationship to explain at this point is one that relates angular motion **($\vec{\Omega}$)** between the ECF system and the vehicle-pointing system. The vehicle-pointing system was created by a rotation θ about the z_1-axis followed by a $-\phi$ rotation about the y_2-axis. Therefore, the angular rate can be written as:

$$\vec{\Omega} = \frac{d\theta}{dt}\hat{e}_{z_1} - \frac{d\phi}{dt}\hat{e}_{y_2} \tag{3}$$

Equation (2) can then be used to write this expression entirely in terms of the vehicle-pointing system:

$$\vec{\Omega} = \left(\frac{d\theta}{dt}\sin\phi\right)\hat{e}_{x_2} - \frac{d\phi}{dt}\hat{e}_{y_2} + \left(\frac{d\theta}{dt}\cos\phi\right)\hat{e}_{z_2} \tag{4}$$

Equations of Motion

By examining Figure (3) one can derive the equation for the velocity, denoted here as ${}^{R}\vec{V}$. The "R" superscript denotes that the velocity is relative to the rotating ECF frame:

$${}^{R}\vec{V} = \left({}^{R}V\sin\gamma\right)\hat{e}_{x_2} + \left({}^{R}V\cos\gamma\cos\psi\right)\hat{e}_{y_2} + \left({}^{R}V\cos\gamma\sin\psi\right)\hat{e}_{z_2} \tag{5}$$

A derivative in ECF frame is related to a derivative in vehicle-pointing reference frame by the following expression

$$\frac{{}^{1}d\vec{r}}{dt} = \frac{{}^{2}d\vec{r}}{dt} + \vec{\omega}_{2/1} \times \vec{r} \tag{6}$$

where $\frac{{}^{1}d\vec{r}}{dt}$ is the velocity relative to the ECF system and $\vec{\omega}_{2/1}$ is the angular rate established in Equation (4) as $\vec{\Omega}$. Therefore, Equation (5) can be rewritten as:

$$ {}^{R}\vec{V} = {}^{2}\frac{d\vec{r}}{dt} + \vec{\Omega} \times \vec{r} \tag{7} $$

After completing the cross-product and writing in terms of the vehicle-pointing system, Equation (7) becomes:

$$ {}^{R}\vec{V} = \dot{r}\ \hat{\mathrm{e}}_{x_2} + \left(r\dot{\theta}\cos\phi\right)\ \hat{\mathrm{e}}_{y_2} + \left(r\dot{\phi}\right)\ \hat{\mathrm{e}}_{z_2} \tag{8} $$

Equations (5) and (8) are both solutions for the relative velocity written in terms of the vehicle-pointing system. Therefore, each component of Equation (5) must be equivalent with its corresponding component in Equation (8):

$$ \dot{r} = {}^{R}V \sin\gamma \tag{9} $$

$$ \dot{\theta} = \frac{{}^{R}V \cos\gamma \cos\psi}{r\cos\phi} \tag{10} $$

$$ \dot{\phi} = \frac{{}^{R}V \cos\gamma \sin\psi}{r} \tag{11} $$

These are known as the three kinematic equations of motion, which define the vehicle's position relative to the rotating planet.

These equations only give half of the information needed to fully describe the motion of the vehicle during reentry. Three more expressions that account for the changes in velocity and direction due to the forces of gravity and aerodynamics are needed.

For the purposes of this research the reentry vehicle is treated as a thrustless point mass. The total force acting on the point mass can then be defined as the summation of the force from gravity and the aerodynamic forces:

$$\vec{F} = \vec{A} + m\vec{g} \tag{12}$$

Another definition of the total force comes from Newton's Second Law where it is assumed the mass is constant and the reference frame is inertial

$$\vec{F} = m\,{}^{I}\frac{d\vec{V}}{dt} \tag{13}$$

where the "I" in the superscript shows that this is a derivative of the velocity in the inertial frame. In order to express the derivative of the velocity in terms of the vehicle-pointing system it is necessary to apply the same technique used to establish the relationship of a derivative in a fixed frame to a derivative in a rotating frame used for Equation (6)

$${}^{I}\frac{d\vec{r}}{dt} = {}^{R}\frac{d\vec{r}}{dt} + \vec{\omega}\times\vec{r} \tag{14}$$

and taking a second derivative to find the derivative of the inertial velocity

$${}^{I}\frac{d^2\vec{r}}{dt^2} = \frac{{}^{I}d\vec{V}}{dt} = {}^{R}\frac{d}{dt}\left({}^{R}\frac{d\vec{r}}{dt} + \vec{\omega}\times\vec{r}\right) + \vec{\omega}\times\left({}^{R}\frac{d\vec{r}}{dt} + \vec{\omega}\times\vec{r}\right) \tag{15}$$

where $\vec{\omega}$ is the angular rate between the ECI and ECF frames. For the purpose of this research, this angular rate can be treated as a constant called $\overrightarrow{\omega_{\oplus}}$. $\frac{{}^{R}d\vec{r}}{dt}$ is the velocity relative to the rotating surface of the planet and $\frac{{}^{R}d^{2}\vec{r}}{dt^{2}}$ is the acceleration relative to the rotating surface of the planet. Equation (15) simplifies to:

$$ {}^{I}\frac{d\vec{V}}{dt} = \frac{d\,{}^{R}V}{dt} + 2\left(\overrightarrow{\omega_{\oplus}} \times {}^{R}V\right) + \overrightarrow{\omega_{\oplus}} \times \left(\overrightarrow{\omega_{\oplus}} \times \vec{r}\right) \tag{16} $$

Replacing ${}^{I}\frac{d\vec{V}}{dt}$ with $\frac{\vec{F}}{m}$ from Equation (13) results in:

$$ m\frac{{}^{R}d\,{}^{R}\vec{V}}{dt} = \vec{F} - 2m\left(\overrightarrow{\omega_{\oplus}} \times {}^{R}\vec{V}\right) - m\left[\overrightarrow{\omega_{\oplus}} \times \left(\overrightarrow{\omega_{\oplus}} \times \vec{r}\right)\right] \tag{17} $$

As defined in the ECF system, $\overrightarrow{\omega_{\oplus}}$ is about the z-axis only. It can be converted to the vehicle-pointing system by applying Equation (2):

$$ \overrightarrow{\omega_{\oplus}} = \left(\omega_{\oplus}\sin\theta\right)\hat{\mathrm{e}}_{x_2} + \left(\omega_{\oplus}\cos\phi\right)\hat{\mathrm{e}}_{z_2} \tag{18} $$

Equations (5) and (18) can then be used to find the cross-product $\overrightarrow{\omega_{\oplus}} \times {}^{R}\vec{V}$ in Equation (17):

$$ \begin{aligned} \overrightarrow{\omega_{\oplus}} \times {}^{R}\vec{V} = &-\left({}^{R}V\omega_{\oplus}\cos\phi\cos\gamma\cos\psi\right)\hat{\mathrm{e}}_{x_2} \\ &+ {}^{R}V\omega_{\oplus}\left(\cos\phi\sin\gamma - \sin\phi\cos\gamma\sin\psi\right)\hat{\mathrm{e}}_{y_2} \\ &+ \left({}^{R}V\omega_{\oplus}\sin\phi\cos\gamma\cos\psi\right)\hat{\mathrm{e}}_{z_2} \end{aligned} \tag{19} $$

Similarly, recognizing that $\vec{r} = r\hat{e}_{x_2}$ the cross product $\overrightarrow{\omega_\oplus} \times \left(\overrightarrow{\omega_\oplus} \times \vec{r}\right)$ can be solved:

$$\overrightarrow{\omega_\oplus} \times \left(\overrightarrow{\omega_\oplus} \times \vec{r}\right) = \left(r\omega_\oplus^2 \cos^2\phi\right)\hat{e}_{x_2} + \left(r\omega_\oplus^2 \sin\phi\cos\phi\right)\hat{e}_{z_2} \tag{20}$$

Recall that the forces ($\vec{F}$) acting on the vehicle are a summation of the aerodynamic forces ($\vec{A}$) and the force due to gravity ($m\vec{g}$). The aerodynamic forces can be further broken down into a drag force ($\vec{D}$) and a lift force ($\vec{L}$). Drag is defined as acting in the direction opposite velocity, and can therefore be described from Figure (3) in the vehicle-pointing system by geometric relation as:

$$\vec{D} = -\left(D\sin\gamma\right)\hat{e}_{x_2} - \left(D\cos\gamma\cos\psi\right)\hat{e}_{y_2} - \left(D\cos\gamma\sin\psi\right)\hat{e}_{z_2} \tag{21}$$

In order to define the lift force a new variable describing the bank angle, σ, must be introduced. σ is defined as the orientation of the lift vector relative to the ($\vec{r}$, ${}^R\vec{V}$) plane as shown in Figure (4).

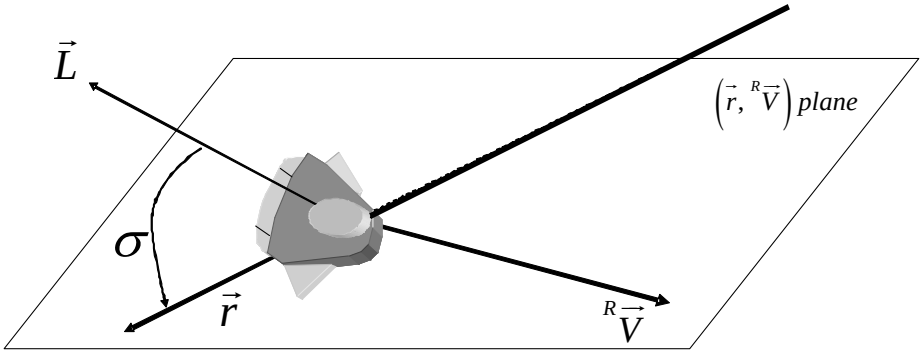

Figure 4: Bank Angle

Recognizing that the lift vector is also perpendicular to the drag vector, lift can then be described in the vehicle-pointing system as a function of bank angle, flight-path angle and heading:

$$\begin{aligned}\vec{L} = {} & L\left(\cos\sigma\cos\gamma\right)\hat{\mathrm{e}}_{x_2} + L(-\cos\sigma\sin\gamma\cos\psi - \sin\sigma\sin\psi)\hat{\mathrm{e}}_{y_2} \\ & + L(-\cos\sigma\sin\gamma\sin\psi + \sin\sigma\cos\psi)\hat{\mathrm{e}}_{z_2}\end{aligned} \tag{22}$$

The final component of the force vector ($\vec{F}$) is the gravity term $m\vec{g}$. Since the bodies in the system are being treated as point masses for the purpose of this research, gravity always acts along the radius vector:

$$m\vec{g} = \left(-mg\right)\hat{\mathrm{e}}_{x_2} \tag{23}$$

Now all of the terms on the right hand side of Equation (17) have been described in the vehicle-pointing system. The equation can be rewritten as:

$$\begin{aligned} m\frac{{}^{R}d\left({}^{R}\vec{V}\right)}{dt} = {} & \left\{\begin{array}{l} +L(\cos\sigma\cos\gamma) - \left(D\sin\gamma\right) - mg \\ -2m\left[-\left({}^{R}V\omega_{\oplus}\cos\phi\cos\gamma\cos\psi\right)\right] \\ -m\left[-\left(r\omega_{\oplus}^{2}\cos^{2}\phi\right)\right] \end{array}\right\}\hat{\mathrm{e}}_{x_2} \\ & + \left\{\begin{array}{l} +L(-\cos\sigma\sin\gamma\cos\psi - \sin\sigma\sin\psi) + D\left(-\cos\gamma\cos\psi\right) \\ -2m\,{}^{R}V\omega_{\oplus}\left(\cos\phi\sin\gamma - \sin\phi\cos\gamma\sin\psi\right) \end{array}\right\}\hat{\mathrm{e}}_{y_2} \\ & + \left\{\begin{array}{l} +L(-\cos\sigma\sin\gamma\sin\psi + \sin\sigma\cos\psi) + D\left(-\cos\gamma\sin\psi\right) \\ -2m\left[\left({}^{R}V\omega_{\oplus}\sin\phi\cos\gamma\cos\psi\right)\right] - m\left[\,r\omega_{\oplus}^{2}\sin\phi\cos\phi\,\right] \end{array}\right\}\hat{\mathrm{e}}_{z_2} \end{aligned} \tag{24}$$

It is now necessary to equate the derivative of the velocity relative to the ECF frame $\left(\frac{{}^{R}d\left({}^{R}\vec{V} \right)}{dt} \right)$ with the derivative of the velocity in the vehicle-pointing system. The same technique used in Equations (6) and (14) is applied here:

$$\frac{{}^{R}d\left({}^{R}\vec{V} \right)}{dt} = \frac{{}^{2}d\left({}^{R}\vec{V} \right)}{dt} + \vec{\Omega} \times {}^{R}\vec{V} \tag{25}$$

where $\vec{\Omega}$ is the angular rate defined in Equation (3). $\frac{{}^{2}d\left({}^{R}\vec{V} \right)}{dt}$ is easily obtained from Equation (5) to yield:

$$\begin{aligned} \frac{{}^{2}d\left({}^{R}\vec{V} \right)}{dt} = & \left({}^{R}\dot{V}\sin\gamma + {}^{R}V\dot{\gamma}\cos\gamma \right)\hat{\mathrm{e}}_{x_2} \\ & + \left({}^{R}\dot{V}\cos\gamma\cos\psi - {}^{R}V\dot{\gamma}\sin\gamma\cos\psi - {}^{R}V\dot{\psi}\cos\gamma\sin\psi \right)\hat{\mathrm{e}}_{y_2} \\ & + \left({}^{R}\dot{V}\cos\gamma\sin\psi - {}^{R}V\dot{\gamma}\sin\gamma\sin\psi + {}^{R}V\dot{\psi}\cos\gamma\cos\psi \right)\hat{\mathrm{e}}_{z_2} \end{aligned} \tag{26}$$

The cross-product $\vec{\Omega} \times {}^{R}\vec{V}$ from Equations (4) and (5) is:

$$\begin{aligned} \vec{\Omega} \times {}^{R}\vec{V} = & {}^{R}V\left(-\dot{\phi}\cos\gamma\sin\psi - \dot{\theta}\cos\phi\cos\gamma\cos\psi \right)\hat{\mathrm{e}}_{x_2} \\ & + {}^{R}V\left(\dot{\theta}\cos\phi\sin\gamma - \dot{\theta}\sin\phi\cos\gamma\sin\psi \right)\hat{\mathrm{e}}_{y_2} \\ & + {}^{R}V\left(\dot{\theta}\sin\phi\cos\gamma\cos\psi + \dot{\phi}\sin\gamma \right)\hat{\mathrm{e}}_{z_2} \end{aligned} \tag{27}$$

Replacing $\dot{\theta}$ and $\dot{\phi}$ with the expressions defined in the kinematic Equations (10) and (11), and combining Equations (25) - (27) gives:

$$\frac{{}^{R}d\left({}^{R}\vec{V}\right)}{dt}=\left({}^{R}\dot{V}\sin\gamma+{}^{R}V\dot{\gamma}\cos\gamma-\frac{{}^{R}V^{2}\cos^{2}\gamma}{r}\right)\hat{\mathrm{e}}_{x_2}$$
$$+\left[\begin{array}{l}{}^{R}\dot{V}\cos\gamma\cos\psi-{}^{R}V\dot{\gamma}\sin\gamma\cos\psi-{}^{R}V\dot{\psi}\cos\gamma\sin\psi\\ \quad+\dfrac{{}^{R}V^{2}}{r}\cos\gamma\cos\psi\left(\sin\gamma-\tan\phi\cos\gamma\sin\psi\right)\end{array}\right]\hat{\mathrm{e}}_{y_2}$$
$$+\left[\begin{array}{l}{}^{R}\dot{V}\cos\gamma\sin\psi-{}^{R}V\dot{\gamma}\sin\gamma\sin\psi+{}^{R}V\dot{\psi}\cos\gamma\cos\psi\\ \quad+\dfrac{{}^{R}V^{2}}{r}\cos\gamma\left(\cos\gamma\cos^{2}\psi\tan\phi+\sin\gamma\sin\psi\right)\end{array}\right]\hat{\mathrm{e}}_{z_2} \tag{28}$$

Equations (24) and (28) are both solutions for the derivative of the velocity in the ECF frame expressed in vehicle-pointing coordinates, and can therefore be equated component-by-component to generate three coupled differential equations. After a bit of work, they can be solved for ${}^{R}\dot{V},\dot{\gamma}$ and $\dot{\psi}$:

$${}^{R}\dot{V}=-\frac{D}{m}-g\sin\gamma+r\omega_{\oplus}^{2}\cos\phi\left(\cos\phi\sin\gamma-\sin\phi\sin\psi\cos\gamma\right) \tag{29}$$

$${}^{R}V\dot{\gamma}=+\frac{L}{m}\cos\sigma-g\cos\gamma+\frac{{}^{R}V^{2}}{r}\cos\gamma+2\,{}^{R}V\omega_{\oplus}\cos\phi\cos\psi$$
$$+r\omega_{\oplus}^{2}\cos\phi\left(\cos\phi\cos\gamma+\sin\phi\sin\psi\sin\gamma\right) \tag{30}$$

$${}^{R}V\dot{\psi}=\frac{L\sin\sigma}{m\cos\gamma}-\frac{{}^{R}V^{2}}{r}\cos\gamma\cos\psi\tan\phi+2\,{}^{R}V\omega_{\oplus}\left(\sin\psi\cos\phi\tan\gamma-\sin\phi\right)$$
$$-\frac{r\omega_{\oplus}^{2}}{\cos\gamma}\sin\phi\cos\phi\cos\psi \tag{31}$$

These are the three differential force equations. When integrated simultaneously with the three kinematic equations, the location and movement of a point mass in a two-body system under the influence of aerodynamic forces can be completely described by the six independent parameters (r, θ, ϕ, ${}^{R}V$, γ and ψ). Note that, just as the velocity (${}^{R}V$) is relative to the rotating atmosphere, so is flight-path angle (γ) and heading (ψ).

The focus of this research is to set up a MATLAB program to integrate these equations and to test their validity and stability at various trajectories.

III. Problem Setup

Overview

This chapter provides details of the practical setup of the problem for a point-mass entering the Earth's atmosphere. It includes definitions of the terms and the methods used to derive the initial values for the equations of motions from ECI coordinate and also from classic orbital elements.

Defining Terms

As stated earlier in the derivation of Equation (16), the angular rate of the Earth's rotation is assumed to be constant. The value for $\omega_{\oplus}$ can then be defined (WGS 84 1998:14):

$$\omega_{\oplus} = 7.292115855228083E-5 \text{ rad/s} \tag{32}$$

The gravity term (g), on the other hand, is not a constant but varies as a function of the radius (Regan, 1984:23)

$$g = g_0 \left(\frac{r_0}{r} \right)^2 \tag{33}$$

where the force of gravity and the radius of the Earth at the equator are the constants g_0 and r_0, respectively. The WGS-84 values for g_0 (Department 2000:3-7) and r_0 (Department 2000:3-2) are:

$$g_0 = 9.7803253359 \text{ m/s}^2 \tag{34}$$

$$r_0 = 6378137\,\text{m} \tag{35}$$

The values for lift (L) and drag (D) are approximated by the following expressions (Hicks, 1993:3-5)

$$D = \frac{\rho S C_D {}^R V^2}{2} \tag{36}$$

$$L = \frac{\rho S C_L {}^R V^2}{2} \tag{37}$$

where S is the reference surface area of the vehicle, C_D and C_L are the coefficients of lift and drag, and ρ is the density of the atmosphere. For this research, the assumption is made that the reference surface area and aerodynamic coefficients are constant and known for the reentry vehicle. The density of the atmosphere varies with altitude. For simplification, this study uses a single layer exponential atmosphere described by (Duncan, 1962:124)

$$\rho = \rho_0 e^{-\beta h} \tag{38}$$

$$\rho_0 = 1.54793\,\text{kg/m}^3 \tag{39}$$

$$\beta = \frac{1}{6935\ \ \text{m}} \tag{40}$$

where ρ_0 is the density of the atmosphere at sea-level, β is the inverse of the scale height for the atmosphere and *h* is the altitude in question. The surface density and scale height are variables by temperature and other factors, but average values were chosen for

simplicity (Hicks, 1993:4-9). This exponential model shows excellent agreement with the 1976 Standard Atmosphere to an altitude of approximately 120 km, beyond which these values are far more difficult to model due to large fluctuations (Regan 1993:39).

Determining the altitude is easy if the Earth is treated as a sphere. It would simply be the difference between the radius (r) to the vehicle and the constant radius of the Earth. But the surface of the Earth is actually an uneven geoid, which is better modeled by an ellipsoid whose ellipticity (*e*) is defined as (Department, 2000:3-7)

$$e = 1 - \frac{r_p}{r_0} \tag{41}$$

$$r_p = 6356752.3142\,\text{m} \tag{42}$$

where r_p is the polar radius of the Earth. The altitude of the vehicle (h), shown in Figure (5), can then be approximated if the assumption is made that

$$r - r_E = h = l \tag{43}$$

where r_E is the radius of the Earth as a function of geocentric latitude. This assumption is valid for the Earth because its ellipticity is very small ($\approx 1/298$) and yields a result accurate to within 2 m for an altitude of 300 km (Regan, 1984:27). Now, a very good approximation for r_E is

$$r_E = \frac{x_a}{\cos\phi} \tag{44}$$

where x_a is the distance shown in Figure (5) and described by (Geocentric 2005):

$$x_a = \frac{r_p}{\sqrt{\tan^2 \phi + (1-e)^2}} \tag{45}$$

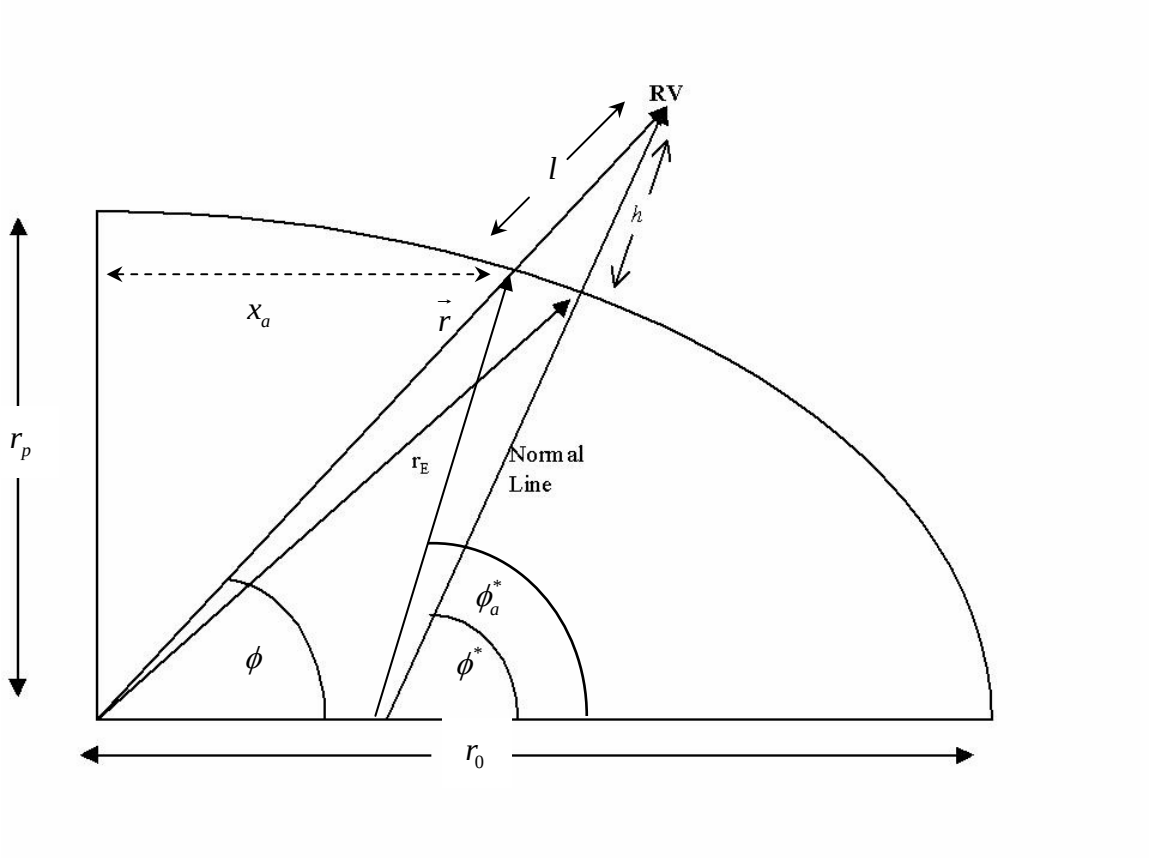

Figure 5. Reference Ellipsoid

Establishing the Initial Conditions

For this study, the differential equations of motion were solved as initial value problems using MATLAB. In order to set the initial conditions, it is essential to define the values needed for the differential equations. In the unlikely case the user knows these particular parameters, they can be entered directly as the initial conditions. If other more

common values are known for the reentry vehicle, conversion to the initial values required for the integration is necessary.

As a means to provide flexibility for the user, the program was written to allow for inputs from three different coordinate systems. First, the user may enter the initial values for the differential equations directly in terms of the vehicle-pointing system coordinates relative to the ECF frame. Second, the values for position and velocity may be defined in terms of the Cartesian ECI frame. Finally, the six classic orbital elements can be used to define the initial reentry conditions. In all cases the user must also define the constants for the reentry vehicle's mass, coefficients of lift and drag, the effective aerodynamic surface area and the constant vehicle bank angle.

Finding Initial Values from Inertial Position and Velocity

If the initial conditions are given in the ECI frame, a method to convert them to the values desired follows. The first step is to calculate the orbital momentum vector ($\vec{H}$) (Weisel, 1997:62):

$$\vec{H} = \vec{r} \times \vec{v} \tag{46}$$

The inertial flight-path angle relative to the local horizontal plane can then be found (Hicks, 2006:17)

$$\cos^{I}\gamma = \frac{H}{rv} \tag{47}$$

which is ambiguous in terms of the sign of the inertial flight-path angle. The flight-path angle is defined as negative when it is Earthward of the local horizontal plane; therefore,

if the vehicle is moving away from the origin of the ECI frame the flight-path angle is positive. In other words, if $\vec{r} \bullet \vec{v} > 0$, the flight-path angle is positive; otherwise, it is negative.

The initial latitude (ϕ) is simple to find due to the fact that the xy-plane in the ECI frame corresponds to the Earth's equator

$$\sin\phi = \frac{z}{r} \tag{48}$$

where *xyz* are the components of *r*. Since $-90^0 \le \phi \le 90^0$, the above equation determines ϕ without ambiguity.

The longitude (θ) is the angle between the position vector projected into the *xy*-plane and the positive *x* direction:

$$\sin\theta = \frac{y}{\sqrt{x^2 + y^2}} \tag{49}$$

This will only yield results of $\pm 90^0$, so a correction can be made by subtracting the result for θ from π when the *x* component of the initial position is negative. Equation (49) is only valid when the ECI and ECF coordinate systems are aligned. Otherwise, a correction must be made for the angular position of the ECF frame relative to the ECI frame:

$$\theta = \sin^{-1}\left(\frac{y}{\sqrt{x^2 + y^2}}\right) - \Delta t\omega \tag{50}$$

The inclination of the trajectory is needed in order to calculate the heading of the reentry vehicle. Inclination is the angle between the angular momentum vector ($\vec{H}$) and the *z*-axis, given by (Weisel, 1997:62):

$$\cos i = \frac{\vec{H} \bullet \hat{e}_z}{H} \tag{51}$$

The inertial heading (${}^I\psi$) is the angle between the projection of the inertial velocity onto the local horizontal plane and the latitude. The inertial heading can be related to the inclination and latitude (Longusky and Vinh, 1980:8):

$$\cos {}^I\psi = \frac{\cos i}{\cos \phi} \tag{52}$$

This result has quadrant ambiguity because the heading can vary a full 360^0. In order to resolve this problem, it is necessary to find the right ascension of the ascending node. First the line of nodes $\vec{n}$ can be found as the vector perpindicular to the angular-momentum vector $\vec{H}$ and the *z*-axis (Weisel, 1997:62):

$$\vec{n} = \frac{\vec{H} \times \hat{e}_z}{\left| \vec{H} \times \hat{e}_z \right|} \tag{53}$$

The line of nodes lies in the equatorial plane and has the form:

$$\vec{n} = \hat{e}_x \cos \Omega + \hat{e}_y \sin \Omega \tag{54}$$

The right ascension of the ascending node (Ω) is now known without ambiguity. The inertial heading is always positive when the object in question is within $\pm 90^0$ of the ascending node. Therefore, when the difference between θ, found in Equation (49), and Ω is less than $\left|\pm 90^0\right|$, the result is $0^0 < {}^I\psi < 180^0$, otherwise $180^0 \le {}^I\psi \le 360^0$.

The next task is to find the heading relative to the rotating atmosphere. For simplification, it is a good idea to define the local horizontal component of the inertial velocity as V_h

$$V_h = V \cos {}^I\gamma \tag{55}$$

and the velocity of the rotating atmosphere (V_e) at the position of the vehicle relative to the center of the Earth (Longusky and- Vinh, 1980:8):

$$V_e = r\omega_\oplus \cos\phi \tag{56}$$

The horizontal component of the relative velocity is then $\vec{V}_h - \vec{V}_e$. The relative heading ψ is the angle between the horizontal component of the relative velocity and the local parallel of latitude as shown in Figure (6). ψ can be found from the law of cosines

$$\cos\psi = \frac{V_h \cos {}^I\psi - V_e}{\sqrt{V_e{}^2 + V_h{}^2 - 2V_eV_h \cos {}^I\psi}} \tag{57}$$

where the quadrant ambiguity of ψ is solved in the same manner as for ${}^I\psi$.

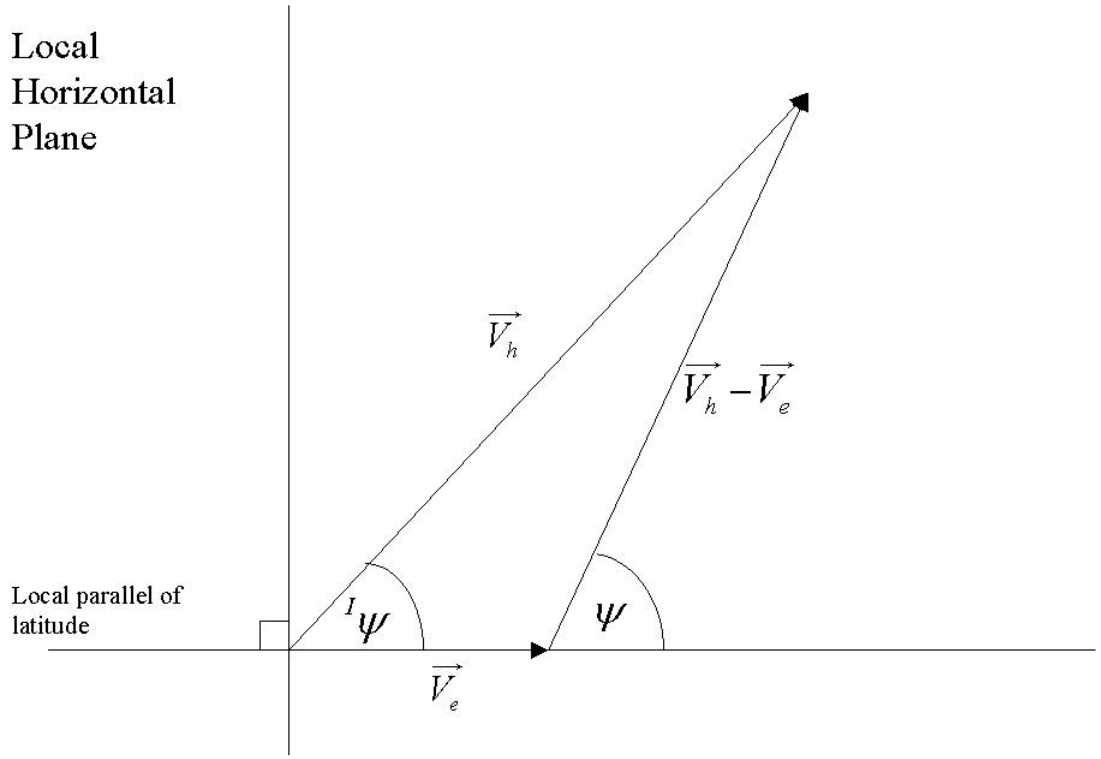

Figure 6. Relationship Between Heading and Velocity

The next value that needs to be defined is the velocity ($^R\vec{V}$) relative to the rotating atmosphere shown in Figure (7). The relative velocity is described as $\vec{V} - \vec{V_e}$ and once again the law of cosines in invoked to come up with a relationship for the magnitude of ^RV:

$$^RV = \sqrt{V^2 + V_e^2 - 2V_eV_h \cos {}^I\psi} \tag{58}$$

As described in Chapter 1 and illustrated in Figure (7), the relative flight path angle, γ, is defined as the angle between the relative velocity and the local horizontal plane. It is determined by the relationship:

$$\sin\gamma = \frac{V \sin {}^{I}\gamma}{{}^{R}V} \tag{60}$$

Since $-90^0 \le \gamma \le 90^0$, Equation (60) determines γ without ambiguity.

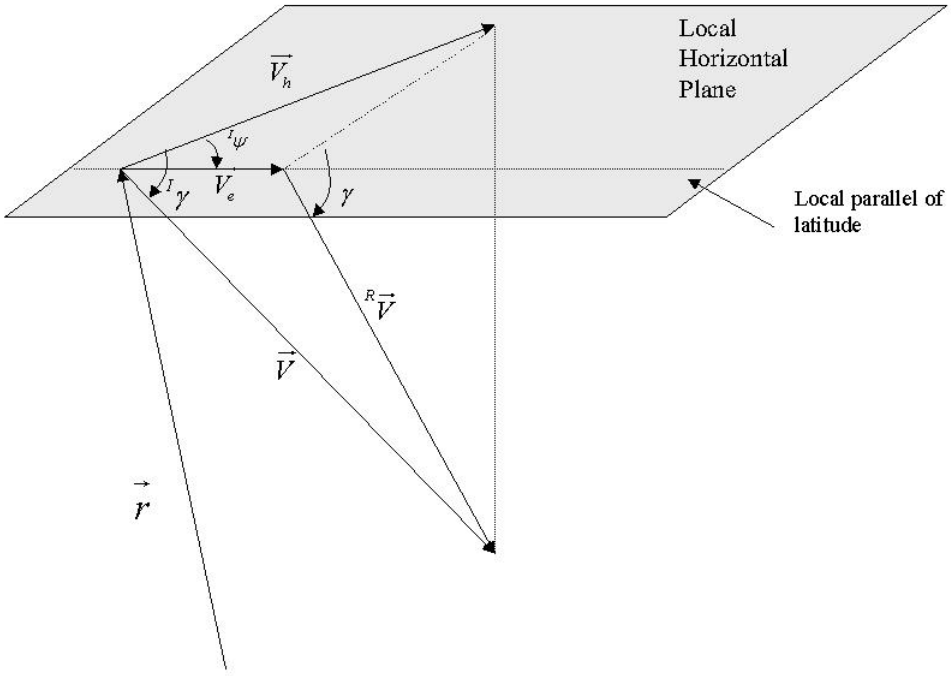

Figure 7. Visualizing Relative Velocity and Flight-path Angle

All of the initial values for the equations of motion $\left(r, \phi, \theta, \psi, {}^{R}V, \gamma\right)$ have now been found and can be integrated to solve for the entry trajectory.

Finding Initial Values from Classic Orbital Elements

As mentioned earlier in this chapter, the initial values for the equations of motion can also be derived from the six classic orbital elements. The semimajor axis (*a)* defines

the size of the orbit and its period. The eccentricity (*e*) determines the orbit's shape. The right ascension of the ascending node (Ω) was addressed in the previous section and identifies the points where the orbital plane intersects the equator, measured as the angle from the vernal equinox to the point where the orbit crosses the equator from south to north. The orbital inclination (*I*) was also discussed in the previous section and is defined as angle between the orbital plane and the equator at the ascending node. The argument of perigee (ω) is the angle from the ascending node to the perigee point. Finally, the true anomaly (*v*) is described as the position of vehicle at a particular instant in time with respect to the argument of perigee (Weisel, 1997:60). Most of these elements are illustrated in Figure (8).

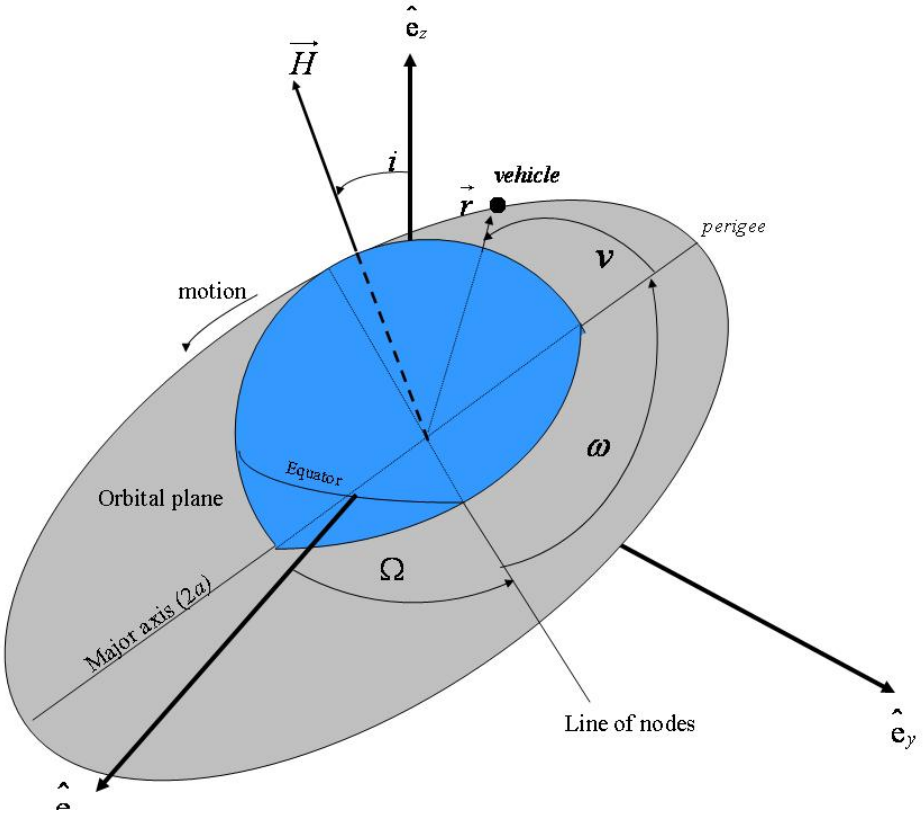

Figure 8. Classic Orbital Elements

As the first step, the scalar radius can be directly calculated from the orbital elements, as shown by Griffin and French (2004:106):

$$r = \frac{a\left(1-e^2\right)}{1+e\cos v} \tag{61}$$

Equation (61) accounts for all possible orbits except parabolic. The radius vector can then be written in the perifocal system, where the center of the Earth is the origin, $\hat{e}_P$ points towards perigee and $\hat{e}_Q$ is in the orbital plane perpendicular to $\hat{e}_P$ as illustrated in Figure (9):

$$\vec{r} = \left(r\cos v\right)\hat{e}_P + \left(r\sin v\right)\hat{e}_Q \tag{62}$$

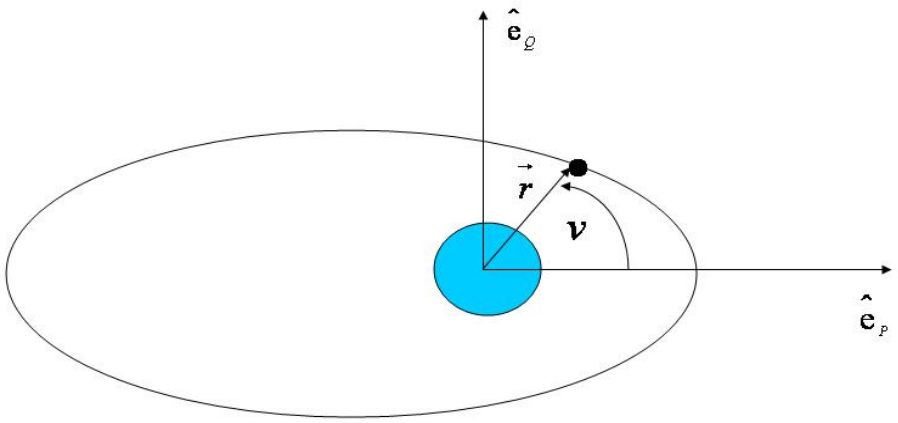

Figure 9. Perifocal Coordinate System

The first step in obtaining the velocity vector is calculating the semi-latus rectum of the orbit (p) (Griffin and French, 2004:107):

$$p = a\left(1-e^2\right) \tag{63}$$

The velocity vector can then be found by (Hicks, 2006:29)

$$\vec{V} = \sqrt{\frac{\mu}{p}}\left[\left(-\sin v\right)\hat{\mathrm{e}}_P + \left(e+\cos v\right)\hat{\mathrm{e}}_Q\right] \tag{64}$$

where μ is the gravitational parameter of the Earth (Weisel, 1997:323):

$$\mu = 3.98601\times10^{14}\,\mathrm{m}^3/\mathrm{s}^2 \tag{65}$$

The values can now be found in the inertial frame by a rotation. The substitutions for the perifocal unit vectors into the ECI frame are given by (Griffin and French, 2004:122):

$$\begin{aligned}\hat{\mathrm{e}}_P &= \left(\cos\Omega\cos\omega-\sin\Omega\sin\omega\cos i\right)\hat{\mathrm{e}}_x \\ &\quad+\left(\sin\Omega\cos\omega+\cos\Omega\sin\omega\cos i\right)\hat{\mathrm{e}}_y+\left(\sin\omega\sin i\right)\hat{\mathrm{e}}_z\end{aligned} \tag{66}$$

$$\begin{aligned}\hat{\mathrm{e}}_Q &= \left(-\cos\Omega\sin\omega-\sin\Omega\cos\omega\cos i\right)\hat{\mathrm{e}}_x \\ &\quad+\left(-\sin\Omega\sin\omega+\cos\Omega\cos\omega\cos i\right)\hat{\mathrm{e}}_y+\left(\cos\omega\sin i\right)\hat{\mathrm{e}}_z\end{aligned} \tag{67}$$

Now that the radius and velocity are known in inertial coordinates, they can be converted to the initial values for the equations of motion as described in the previous section.

IV. Program Operation

Overview

This chapter is a brief description of the operation of the MATLAB reentry software developed in this research. First, a description of the user operations is presented along with graphical illustrations of the interfaces. Next, the software system processes are outlined and presented in the form of a flow chart.

Operation

To begin operation of the reentry software, the user must ensure that the folder containing the program *reentry.m* is opened in MATLAB's Current Directory. Typing "reentry" at the command line will then initiate the program by bringing up a graphical user interface (GUI) titled "Reentry Program Initiation" as shown in Figure (10). This GUI allows the operator to select the desired coordinate system of the initial conditions as

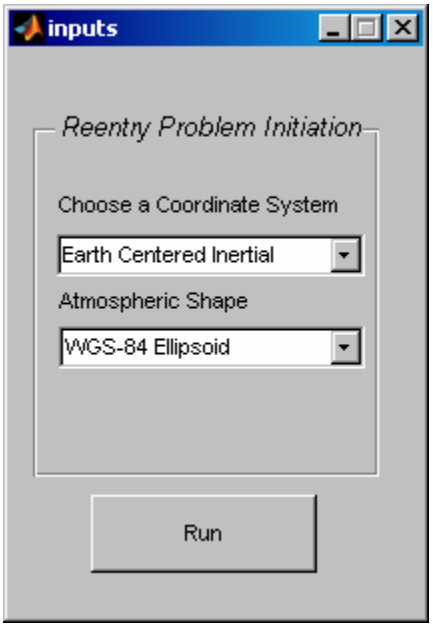

Figure 10. GUI for Reentry Problem Initiation

well as the preferred shape of the atmosphere from drop-down menus listing the options. Initial conditions can be in the vehicle pointing system, the ECI system, or described by the classic orbital elements. The atmosphere may be defined as spherical or conforming to the WGS-84 ellipsoidal approximation for the Earth's shape.

Clicking "Run" on the initiation GUI will bring up a second GUI, as shown in Figure (11), to allow the user to input the vehicle parameters and initial conditions of reentry. The operator simply needs to include a value in each cell corresponding to each particular parameter. Clicking "Run" initiates the simulation.

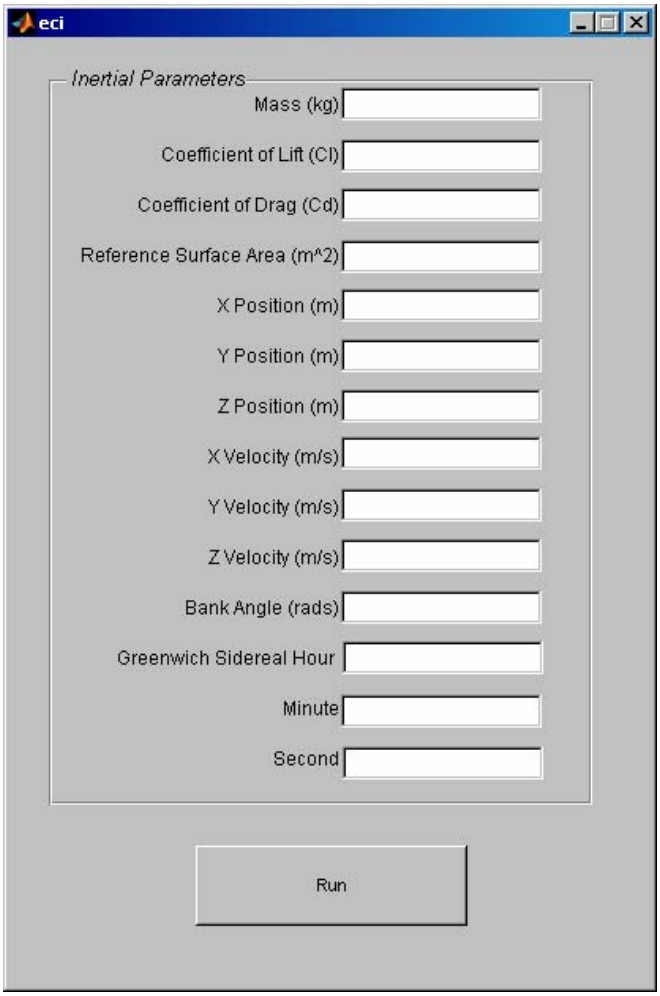

Figure 11. GUI for Vehicle Parameters and Initial Conditions

The software then automatically generates a plot, like that shown in Figure (12), for each of the parameters solved for in the six equations of motion; altitude, velocity, flight-path angle, geocentric latitude, longitude, and heading. It also generates a plot showing the acceleration experienced by the vehicle.

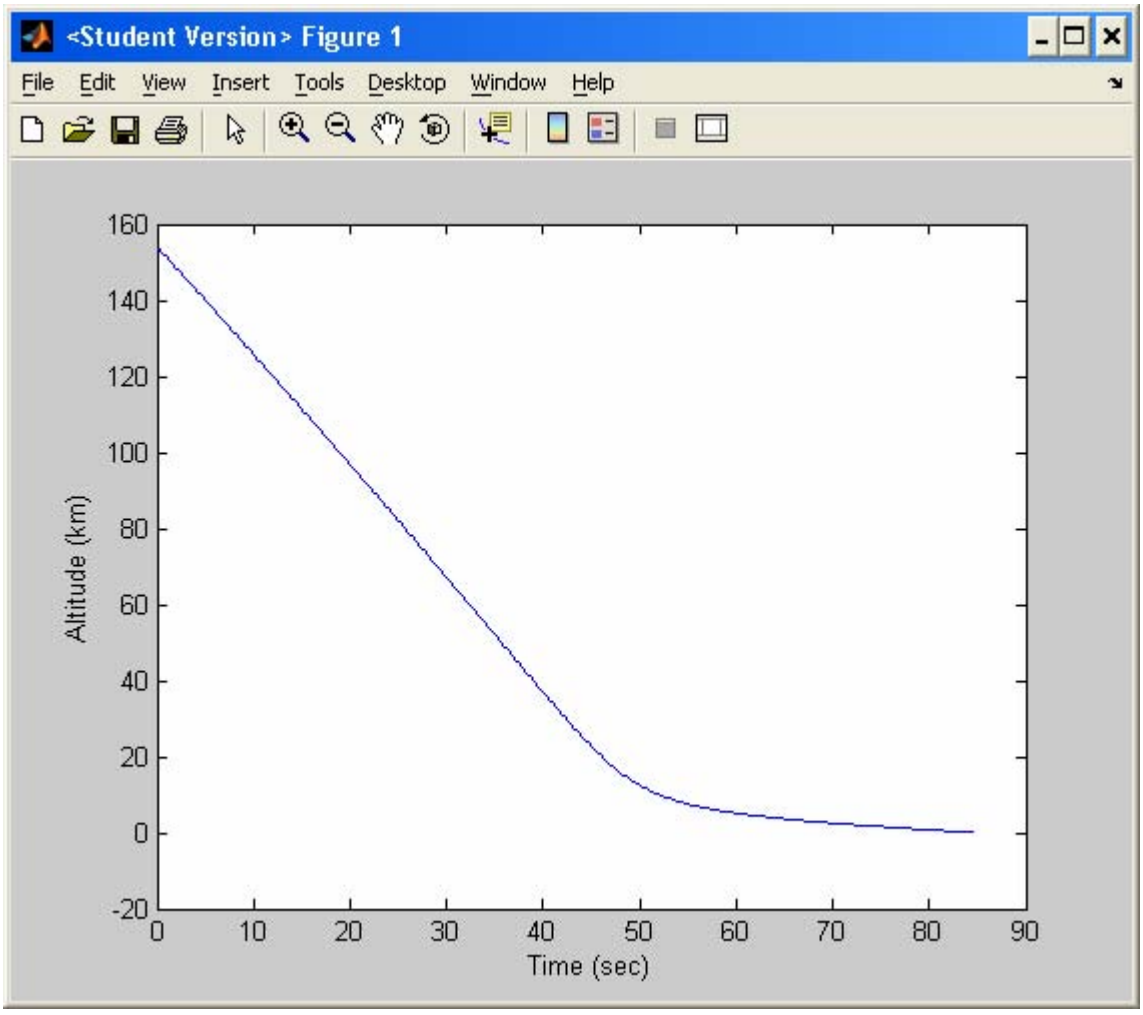

Figure 13. Sample Plot of Results

Additionally, the results are tabulated and saved into a matrix named *Y*. The *Y* matrix can be selected from the MATLAB Workspace and displayed in the Array Editor. *Y* has six columns that are values for radius (m), relative velocity (m/s), relative flight-

path angle (rads), longitude (rads), latitude (rads), and relative heading (rads), respectively. Each row of *Y* corresponds to a time from the vector *t*, which is also found in the Workspace. *t* is composed of the time, in seconds, for each step of the integration, starting at zero for the initial conditions.

The vehicle parameters and initial conditions are automatically displayed as a matrix call "user_inputs" in the MATLAB Array Editor along with a matrix of their description called "parameters". The initial conditions displayed in "user_inputs" can be modified directly in the Array Editor and the entire simulation can be run again by typing "reentry" at the MATLAB Command Line. Subsequent initiations of the program in this manner will not bring up the GUIs described earlier. In order to change the coordinate system or atmosphere shape, the MATLAB Workspace needs to be cleared and the program re-initiated.

Software System Process

The MATLAB codes developed in this research, along with a description of each code's function, is presented in detail in the Appendix. The codes themselves contain comments describing the purpose of every section. A general overview of the software system process is shown here in Figures (13) and (14).

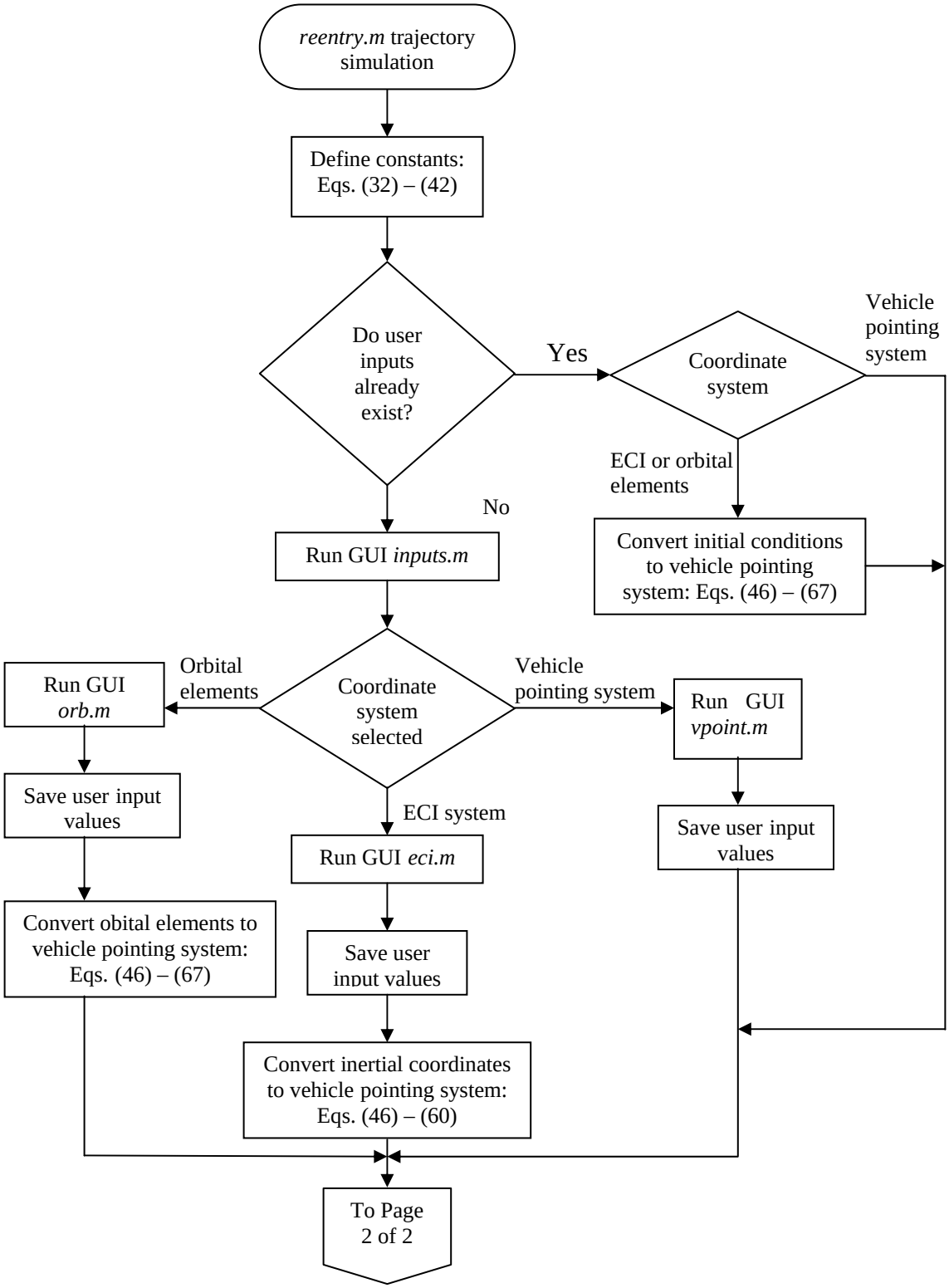

Figure 13. Software System Process Chart Page 1 of 2

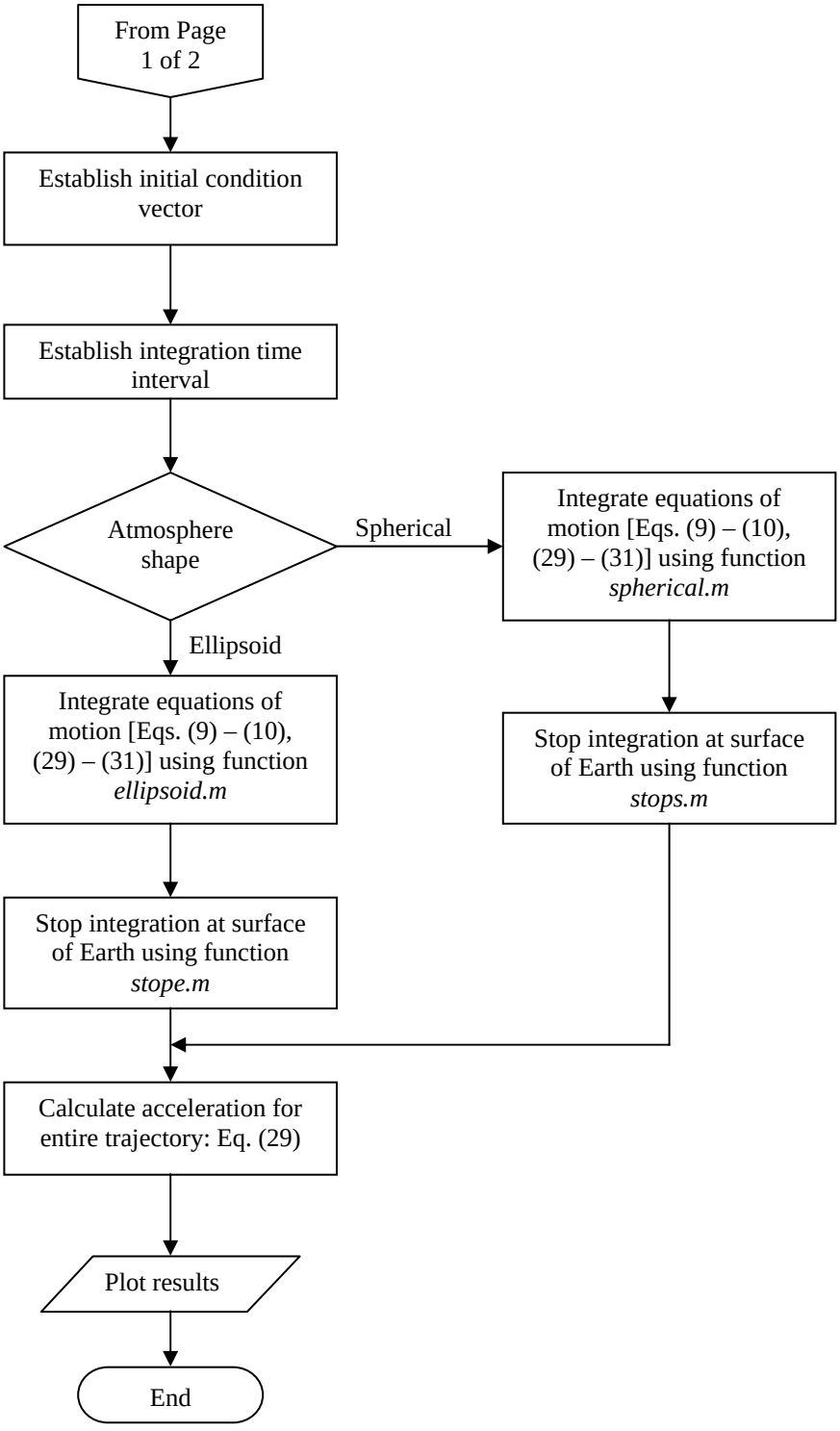

Figure 14. Software System Process Chart Page 2 of 2

V. Program Validation

Overview

This chapter shows comparisons of the results obtained from the MATLAB reentry program with those obtained from other established software programs as well as the results that are predicted by first-order analytical approximations of the equations of motion. First, the software will be compared with two trajectories generated by TARGET. Next, it will be compared to a trajectory generated by IMPULSE. The MATLAB program will then be tested against the first-order solutions at near-circular orbital velocities for a shallow gliding entry, a steep gliding entry, and a steep ballistic entry. Finally, the program will be examined against the first-order approximation for medium gliding entry at supercircular orbital velocity.

Comparison with TARGET

As mentioned in the background for this research, TARGET was developed as a tool for initial analysis of entry trajectories. It was intended to be used as a substitute for POST during the systems concept stage when simplicity and speed is preferred and a good estimate will suffice. As such, TARGET was validated against POST and the trajectories they produced agreed quite nicely. TARGET results only differed from POST by less than .004% for flight-path angle, .001% for latitude and longitude, and .6% for velocity (Hicks, 1993:4-7). By validating the MATLAB program results against TARGET, it is also implicitly validated against POST.

Case #1.

The first comparison consists of the trajectory for a notional high-speed, ballistic projectile. Both TARGET and the MATLAB program begin with the same initial parameters as shown in Table (1). For this case the classic orbital elements of the reentry vehicle are used as initial conditions so the function of the MATLAB program to convert these elements as described in Chapter III is tested. TARGET assumes the Earth's atmosphere is spherical, therefore the MATLAB program used a spherical atmosphere to remain consistent. The trajectories generated by each program are plotted in Figures (10) - (15).

Table (1): Parameters for Comparison with TARGET case #1

Mass:	13 kg
C_L:	0.00
C_D:	.08575
S:	.036609615 m^2
$a(0)$:	-1840738 m
$\nu(0)$:	4.552029 rad
$i(0)$:	1.178101
$e(0)$:	1.889431
$\omega(0)$:	2.377746 rad
$\Omega(0)$:	4.213107 rad
σ:	0 rad
Corresponding initial conditions for equations of motion (automatically generated)	
$r(0)$:	6774400 m
$\theta(0)$:	4.4943 rad
$\phi(0)$:	.5903 rad
${}^{R}V(0)$:	18220 m/s
$\gamma(0)$:	-1.2217 rad
$\psi(0)$:	1.1506 rad

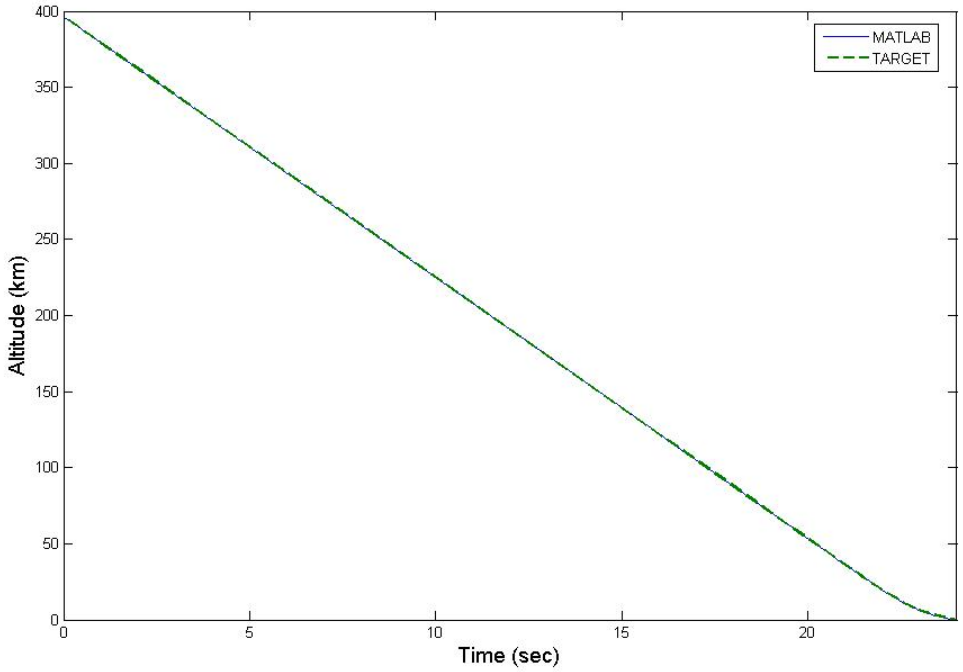

Figure 15. Altitude/Time Comparison for MATLAB/TARGET Case #1

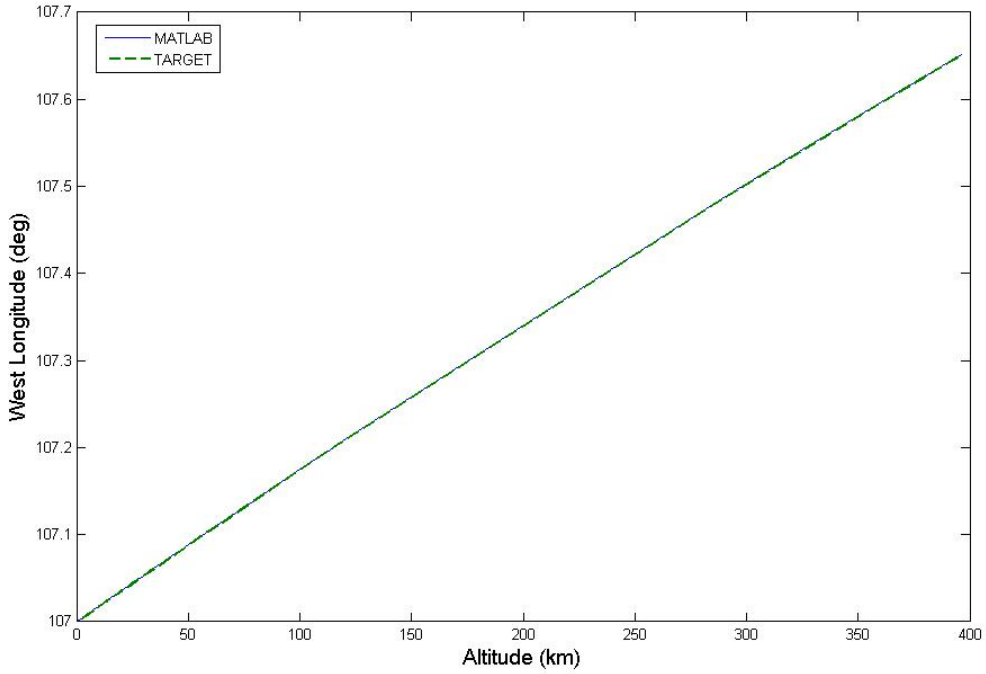

Figure 16. Longitude/Altitude Comparison for MATLAB/TARGET Case #1

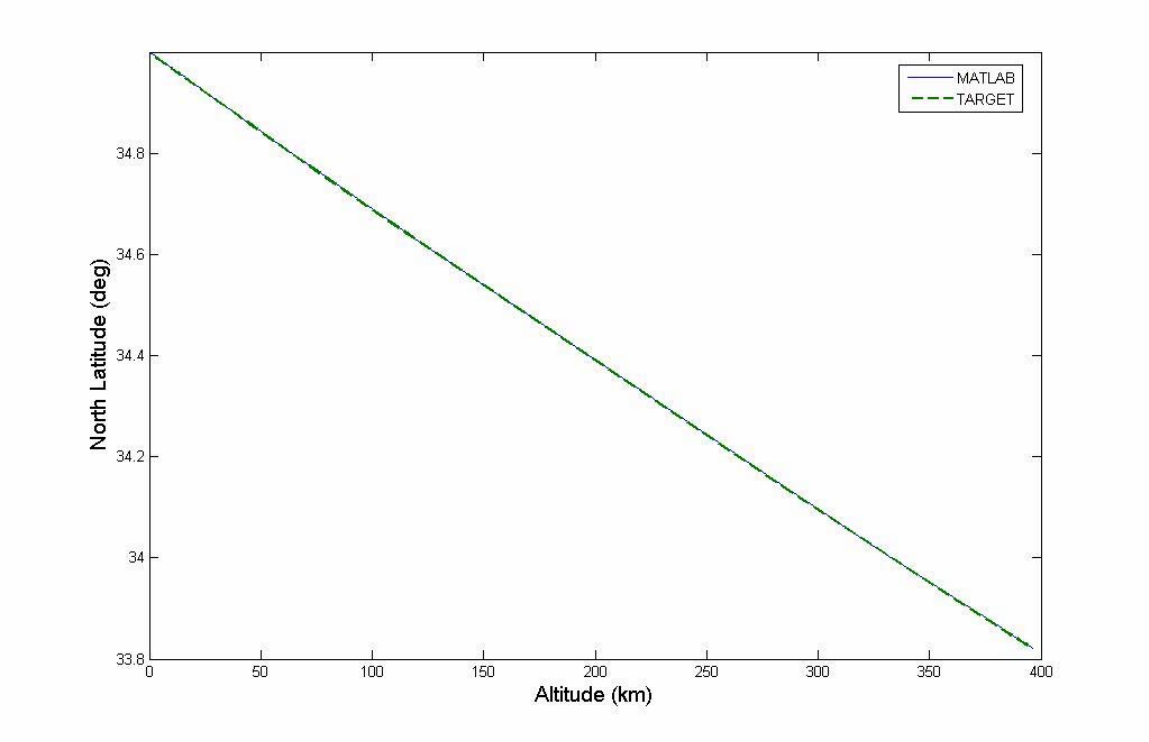

Figure 17. Latitude/Altitude Comparison for MATLAB/TARGET Case #1

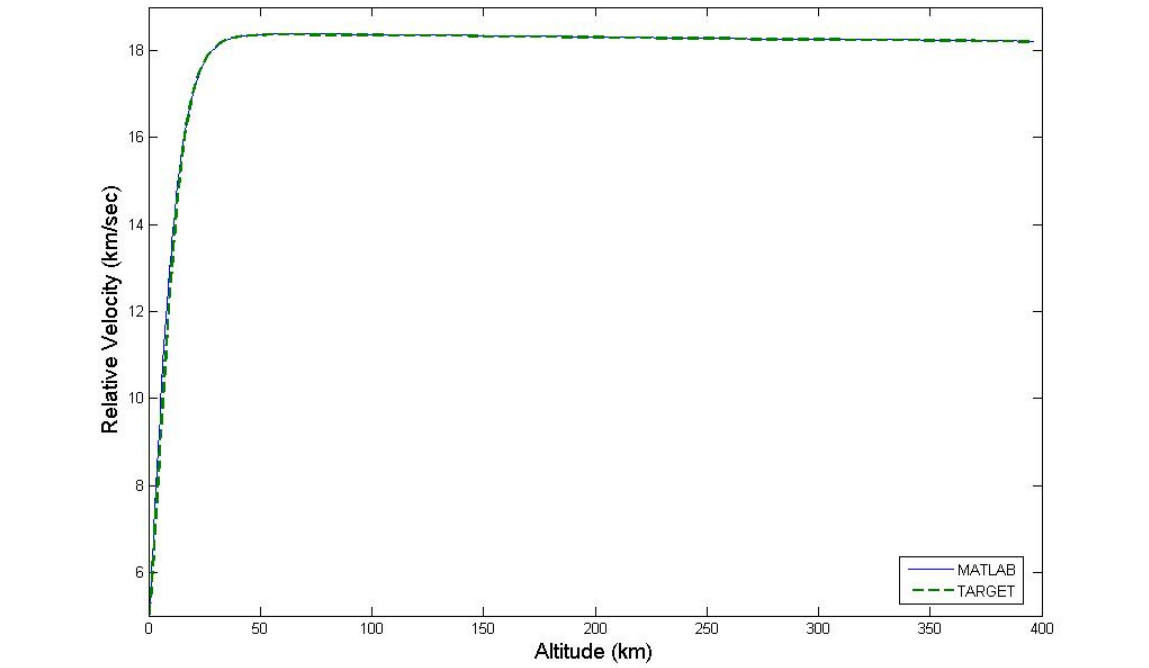

Figure 18. Velocity/Altitude Comparison for MATLAB/TARGET Case #1

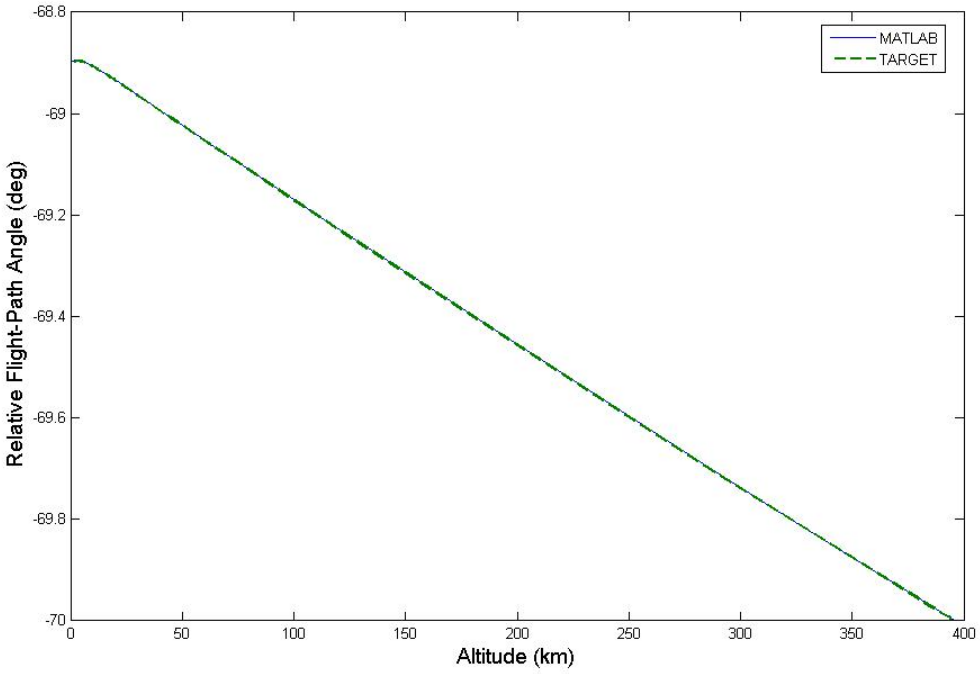

Figure 19. Flight-Path Angle/Altitude Comparison for MATLAB/TARGET Case #1

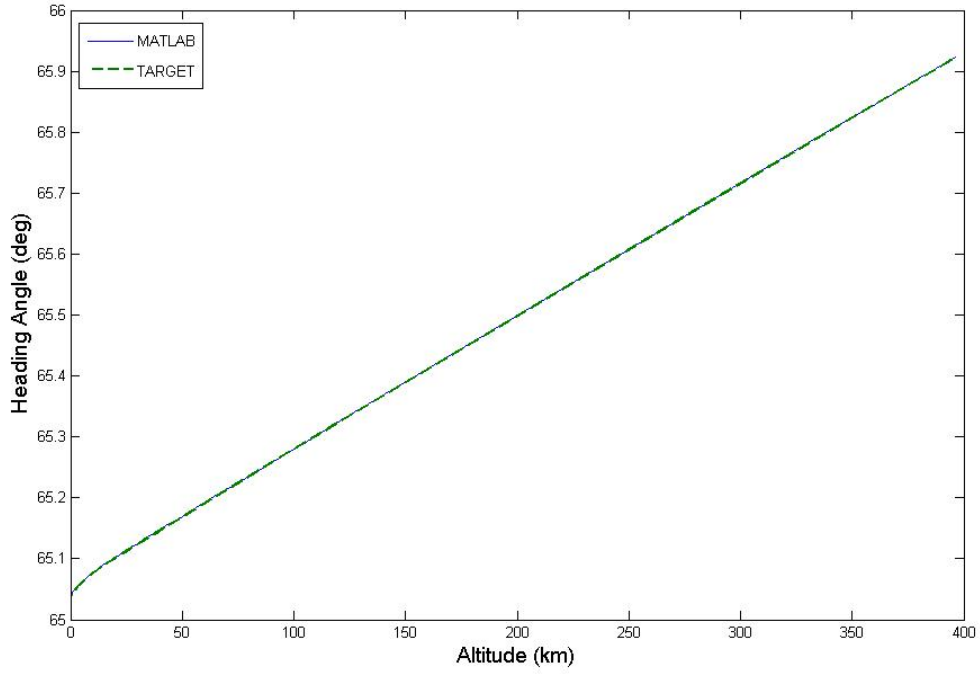

Figure 20. Heading/Altitude Comparison for MATLAB/TARGET Case #1

In order to quantify the results of the comparison, the differences were summarized on a percentage basis and plotted against time in Figures (16) and (17). The percentage values were computed at each interval of time using:

$$\% \text{ difference} = \frac{(\text{TARGET Result}) - (\text{MATLAB Result})}{\text{MATLAB Result}} \tag{68}$$

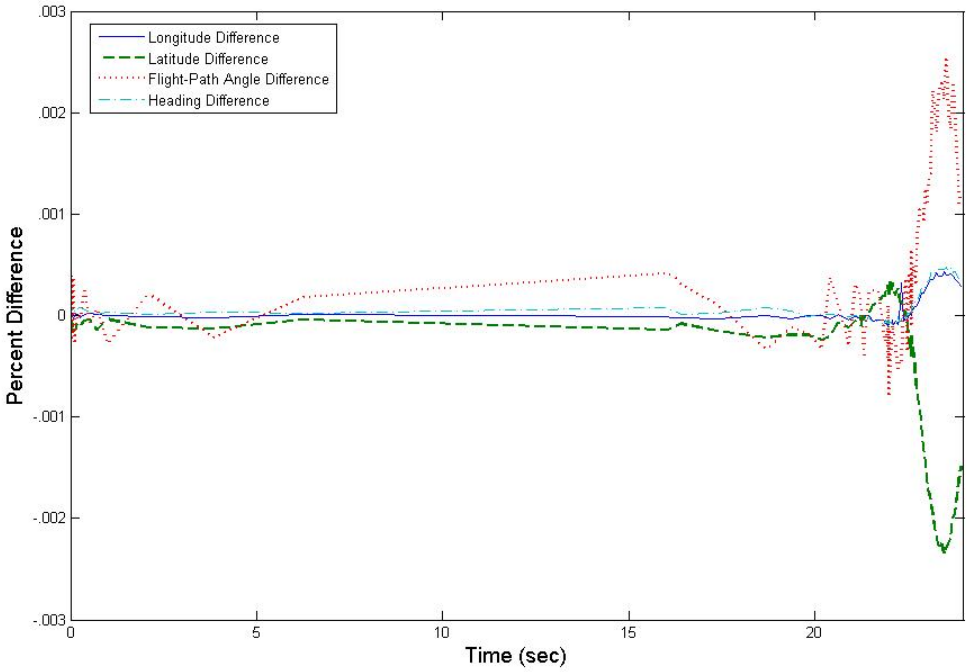

Figure 21. Comparison of Angular Data for MATLAB/TARGET Case #1

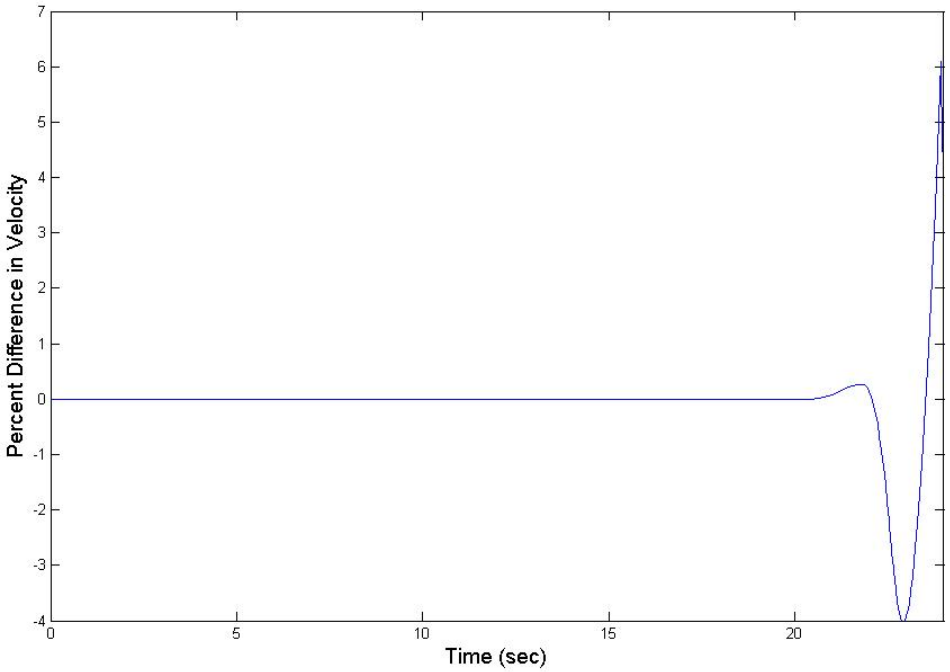

Figure 22. Comparison of Velocities for MATLAB/TARGET Case #1

These results illustrate that the MATLAB program and TARGET produce comparable results when starting with the same initial parameters. It also demonstrates the validity of the routine used in MATLAB to convert the orbital elements to the parameters used in the equations of motion. The deviation in velocity at the lower levels of the atmosphere is almost certainly a result of the different atmospheric models used by the two programs. TARGET uses a more complex seven-layered atmosphere while the MATLAB program uses a simplified single layer exponential atmosphere.

Case #2.

A second comparison was performed between the MATLAB routine and TARGET using the vehicle tested by Albrecht in her analysis of the effects of lift and thrust on maneuvering a reentry vehicle (2005:24-29). This vehicle is a much larger and heavier model than the one in the previous test and includes a substantial amount of lift at

a significant bank angle. It also begins its trajectory at a much lower velocity. The initial parameters are listed in Table (2) and the results are shown in Figures (18) - (26).

Table (2): Parameters for Comparison with TARGET case #2

Mass:	700 kg
C_L:	.2
C_D:	.2
S:	.216419 m^2
$r(0)$:	6500140 m
$\theta(0)$:	-1.34732 rad
$\phi(0)$:	.65176 rad
${}^{R}V(0)$:	3007.2 m/s
$\gamma(0)$:	-.523599 rad
$\psi(0)$:	1.195427 rad
σ:	.8 rad

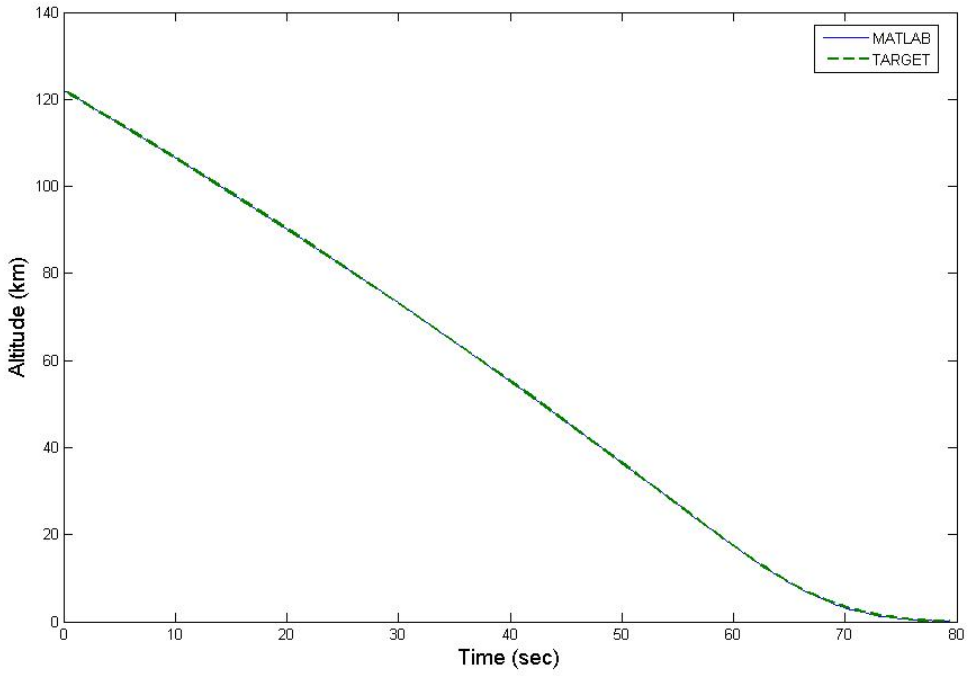

Figure 23. Altitude/Time comparison for MATLAB/TARGET Case #2

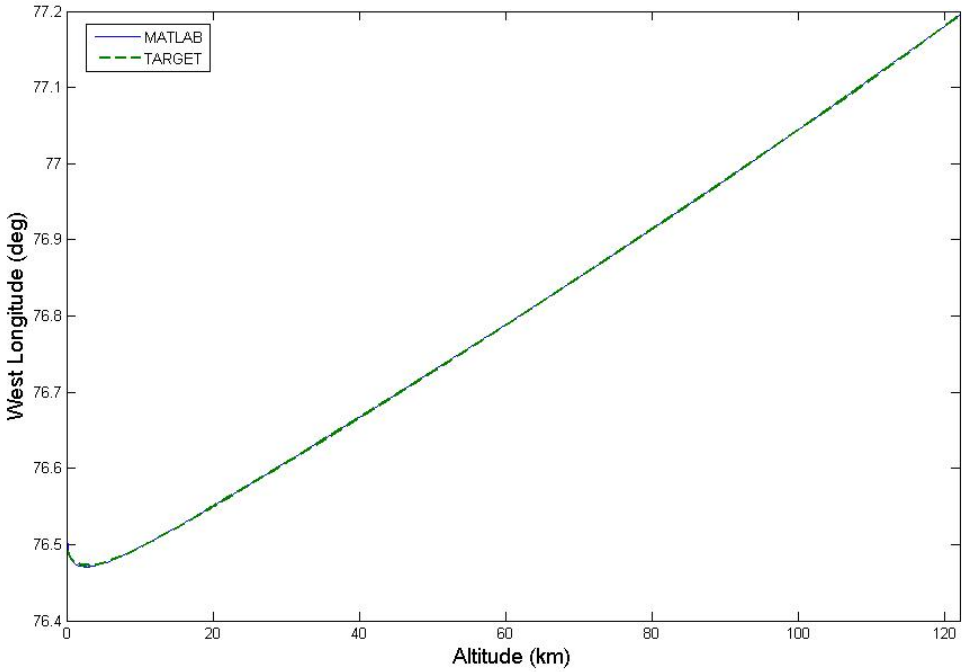

Figure 24. Longitude/Altitude Comparison for MATLAB/TARGET Case #2

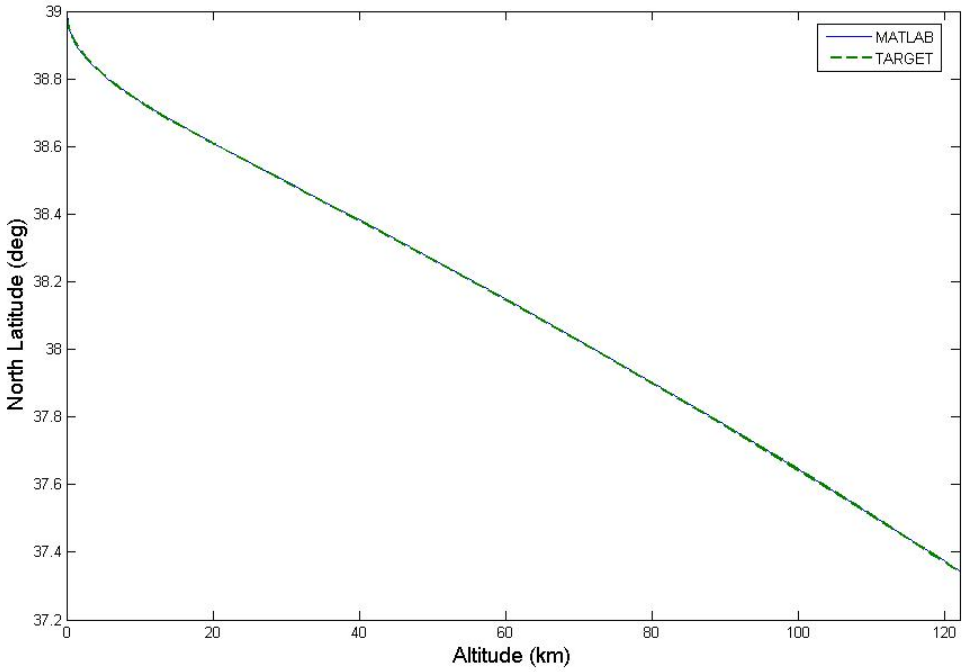

Figure 25. Latitude/Altitude Comparison for MATLAB/TARGET Case #2

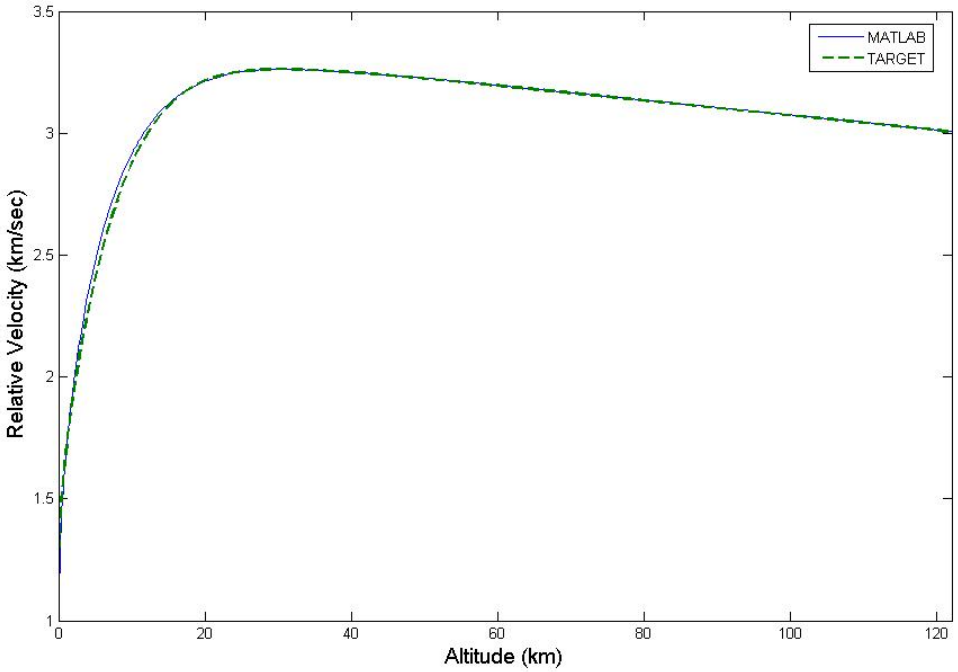

Figure 26. Velocity/Altitude Comparison for MATLAB/TARGET Case #2

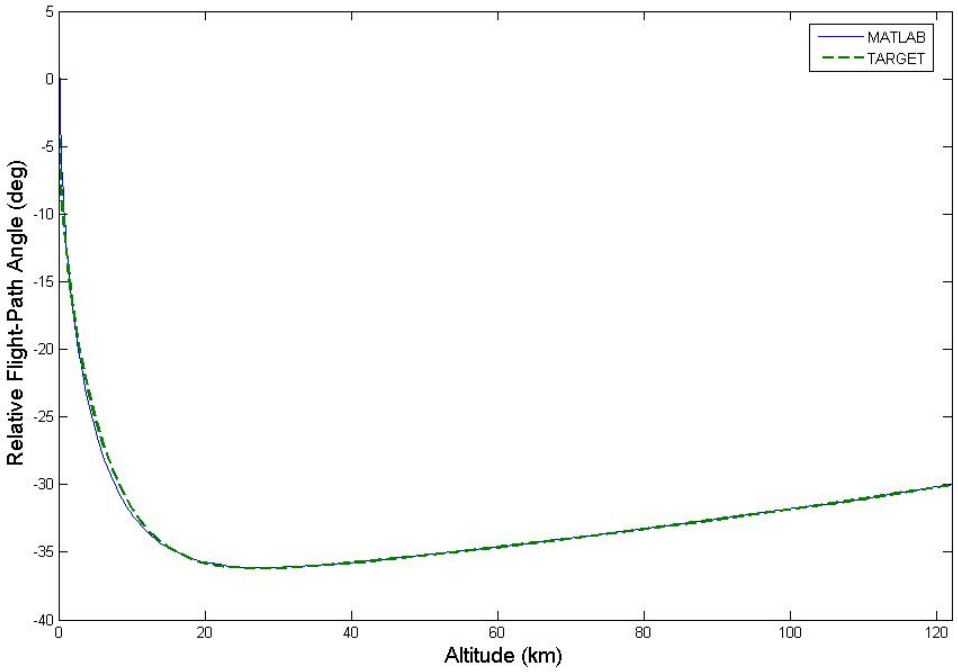

Figure 27. Flight-Path Angle/Altitude Comparison for MATLAB/TARGET Case #2

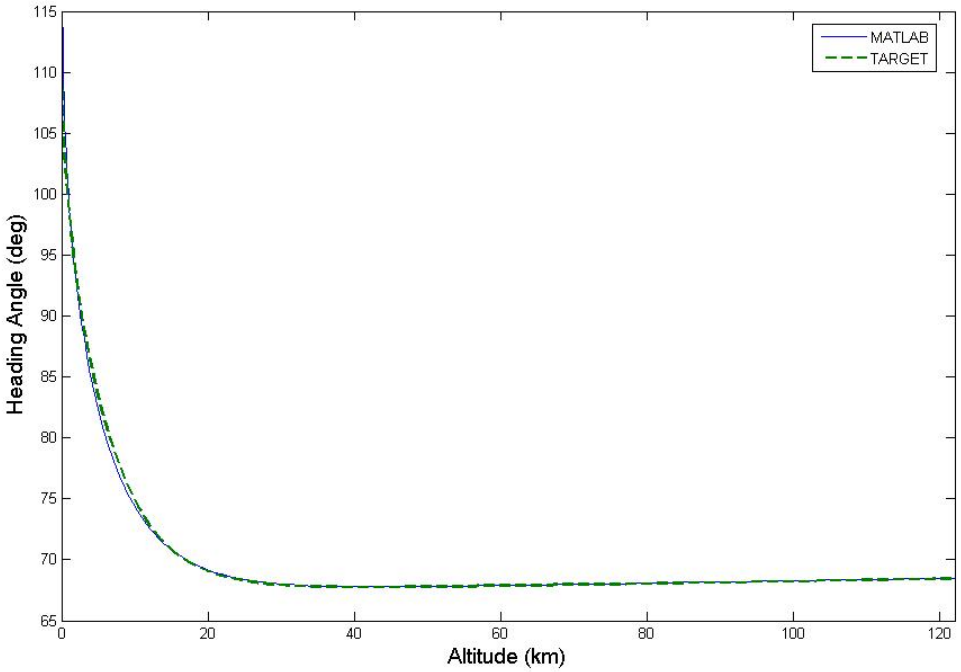

Figure 28. Heading/Altitude Comparison for MATLAB/TARGET Case #2

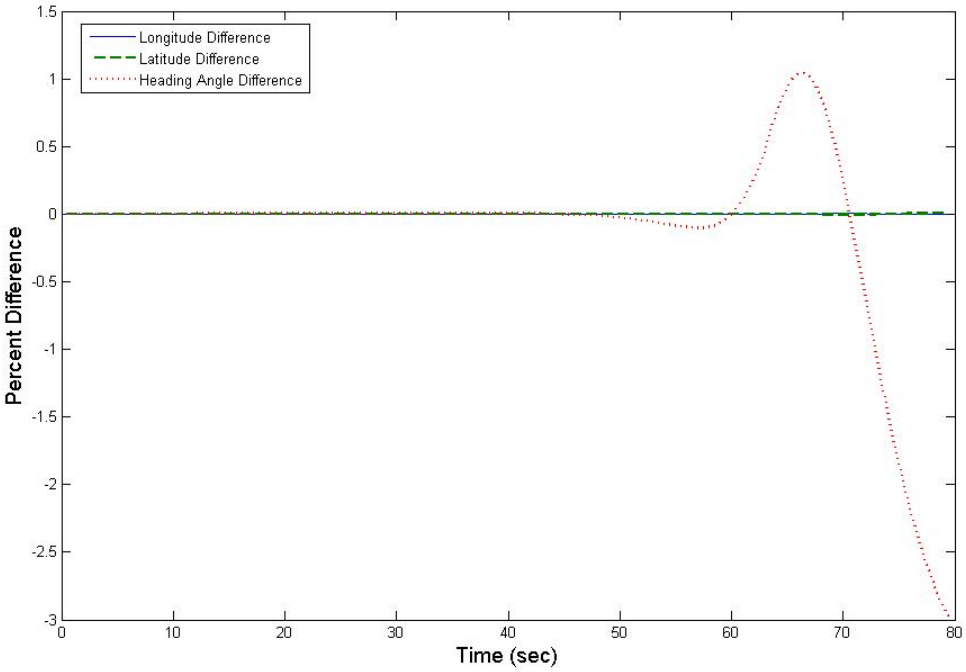

Figure 29. Comparison of Angular Data for MATLAB/TARGET Case #2

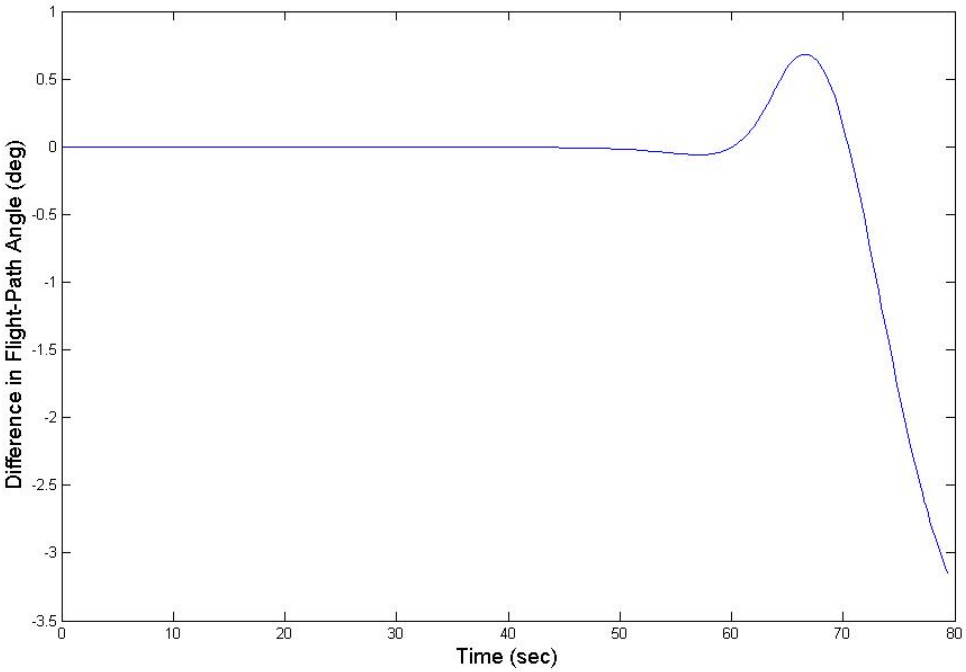

Figure 30. Comparison of Flight-Path Angles for MATLAB/TARGET Case #2

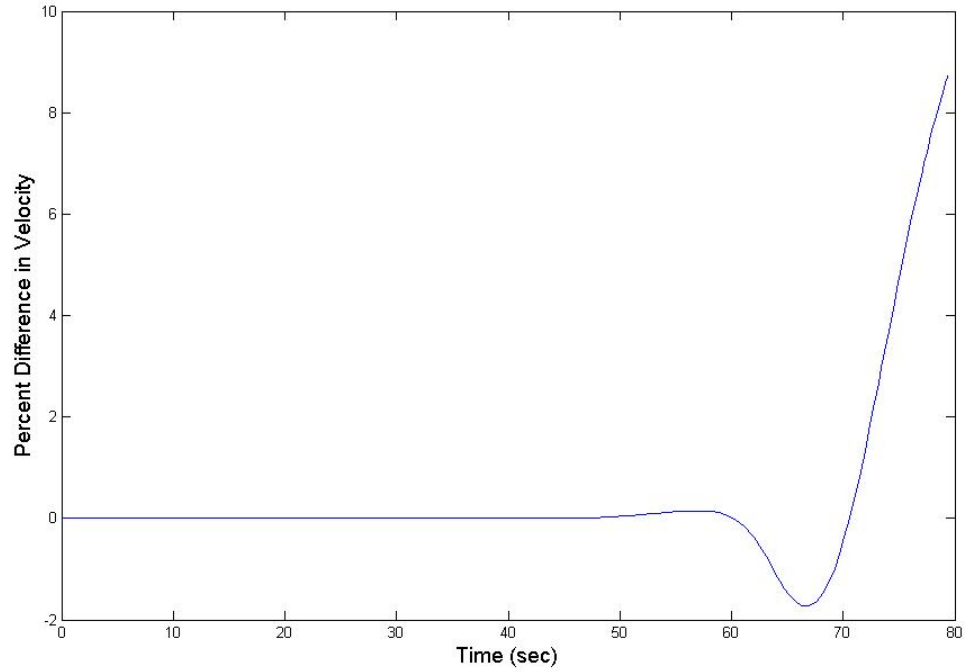

Figure 31. Comparison of Velocities for MATLAB/TARGET Case #2

These results demonstrate the ability of the MATLAB routine to competently simulate modeled trajectories for vehicles with lift and a bank angle. Once again the results for predicted position from the MATLAB program and TARGET are virtually identical. While not excessive, there are deviations in the predicted velocity, flight-path angle and heading angle between the two programs. This is probably attributable to the different atmospheric models used. The fact that Case #2 was a high-lift model while Case #1 was a ballistic model is almost certainly the reason those deviations were more pronounced in this simulation than in the previous one.

Comparison with IMPULSE

The MATLAB program was also compared to IMPULSE to determine if they produced similar results. As described in the Introduction, IMPULSE is designed to simulate the trajectories of ballistic missiles from launch to impact. Since the MATLAB program does not consider thrust, the comparison is made starting from the point immediately following burnout and separation of the reentry vehicle.

One problem in developing the comparison between the two programs is that the output from IMPULSE displays the latitude in geodetic coordinates rather the geocentric coordinates used in Equations (9) – (11) and (29) – (31). Therefore, the geocentric latitude output from the MATLAB program was converted to geodetic latitude for comparison purposes (Geocentric 2005).

To begin with, the angle shown as ϕ_a^* in Figure (27) is the relationship between the geocentric latitude and the geodetic latitude at the point where $\vec{r}$ crosses the Earth's surface, described by

$$\tan\phi_a^* = \frac{\tan\phi}{(1-e)^2} \tag{69}$$

where *e* is the ellipticity of the Earth described in Equation (41). The angular difference between geocentric and geodetic latitude at this point on the Earth is:

$$\delta\phi = \phi_a^* - \phi \tag{70}$$

The radius of curvature of the Earth (ρ_a) at this point in the Meridian is:

$$\rho_a = \frac{r_0(1-e)^2}{\left(1-\left(2e-e^2\right)\sin^2\phi_a^*\right)^{\frac{3}{2}}} \tag{71}$$

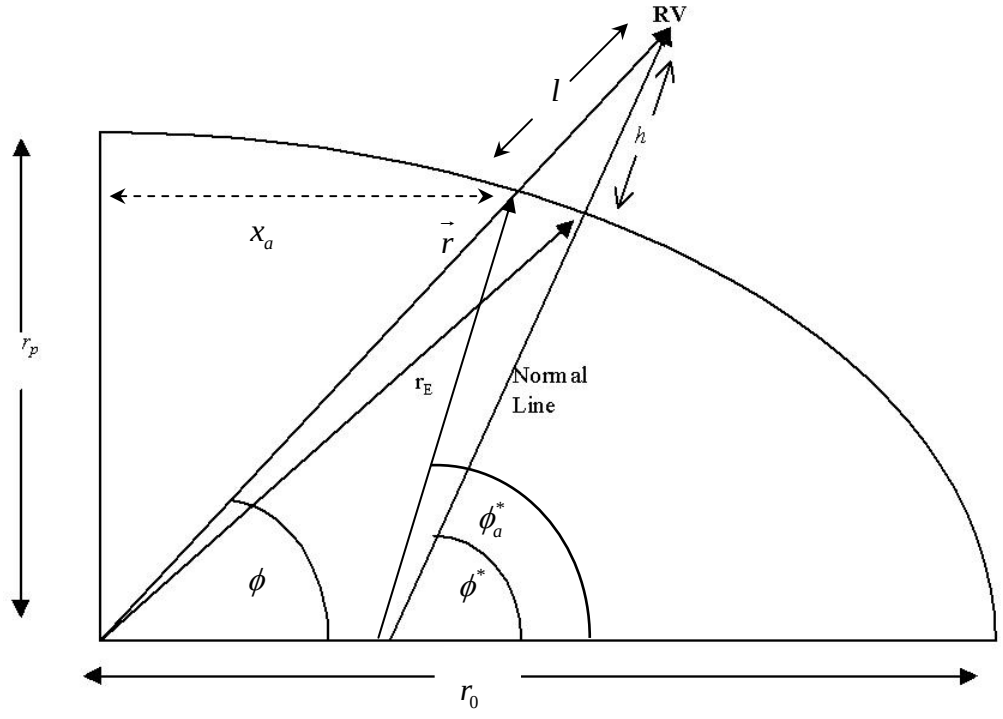

Figure 32. Reference Ellipsoid

The difference between ϕ_a^* and the geodetic latitude of the reentry vehicle (ϕ^*) is then approximated as a function of the height above the surface of the planet where *h* and *l* in Figure (5) are assumed to be equal for the Earth as expressed in Equation (43):

$$\tan \delta\phi^* = \frac{h \sin \delta\phi}{\rho_a + h} \tag{72}$$

Subtracting $\delta\phi^*$ from ϕ_a^* gives the geodetic latitude:

$$\phi^* = \phi_a^* - \delta\phi^* \tag{73}$$

The vehicle parameters and the conditions following burnout, described in the ECI frame, are shown in Table (3). The results of the comparison are shown in Figures (28) – (35).

Table (3): Parameters for Comparison with IMPULSE

Mass:	1088.64 kg	Corresponding initial conditions for equations of motion (automatically generated)	
C_L:	0.00		
C_D:	.11	$r(0)$:	6946000 m
S:	1.880022 m^2	$\theta(0)$:	-1.374 rad
X(0):	1044303.78549 m	$\phi(0)$:	.772 rad
Y(0):	-4866124.54174 m	${}^{R}V(0)$:	5194.5 m/s
Z(0):	4845298.63761 m	$\gamma(0)$:	1.0775 rad
$\dot{X}(0)$:	-1094.1974 m/s	$\psi(0)$:	-2.7926
$\dot{Y}(0)$:	-4187.85837 m/s		
$\dot{Z}(0)$:	2588.77299 m/s		
σ:	0 rad		
t(0):	00:03:20.6 (sidereal time)		

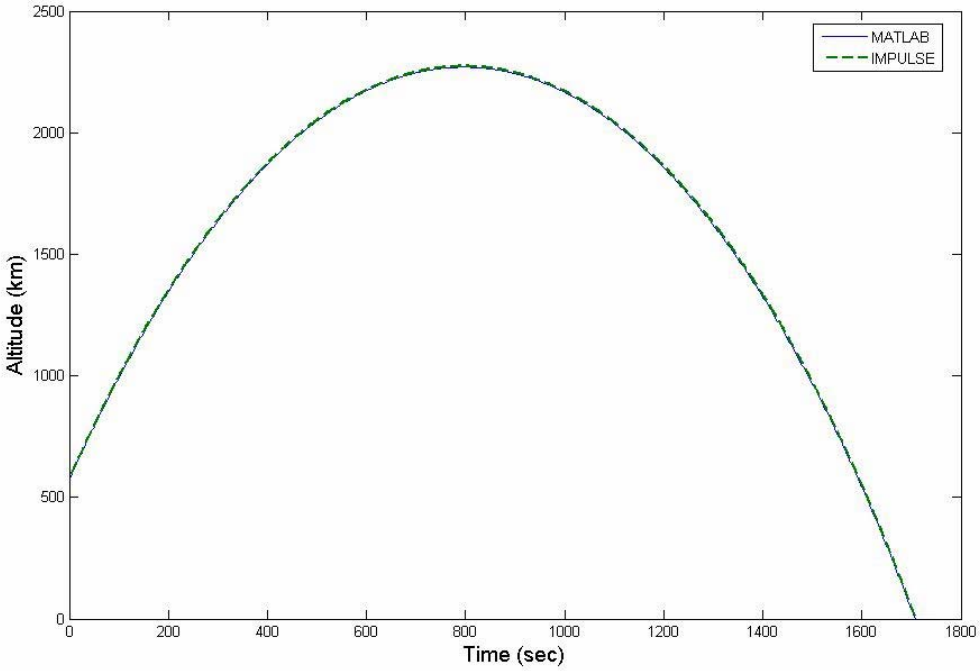

Figure 33. Altitude/Time Comparison for MATLAB/IMPULSE

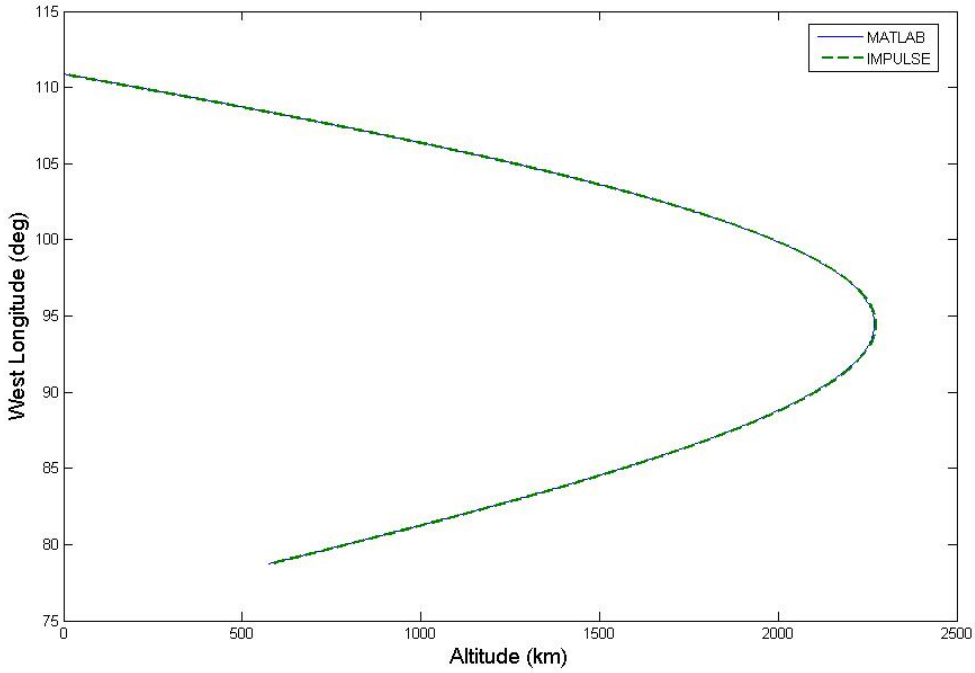

Figure 34. Longitude/Altitude Comparison for MATLAB/IMPULSE

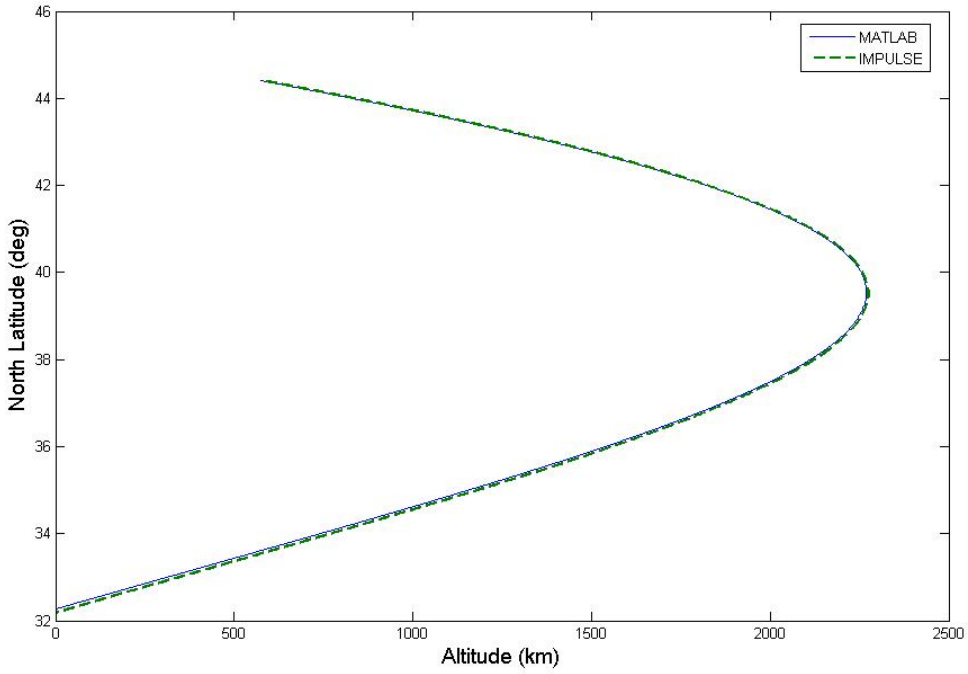

Figure 35. Geodetic Latitude/Altitude Comparison for MATLAB/IMPULSE

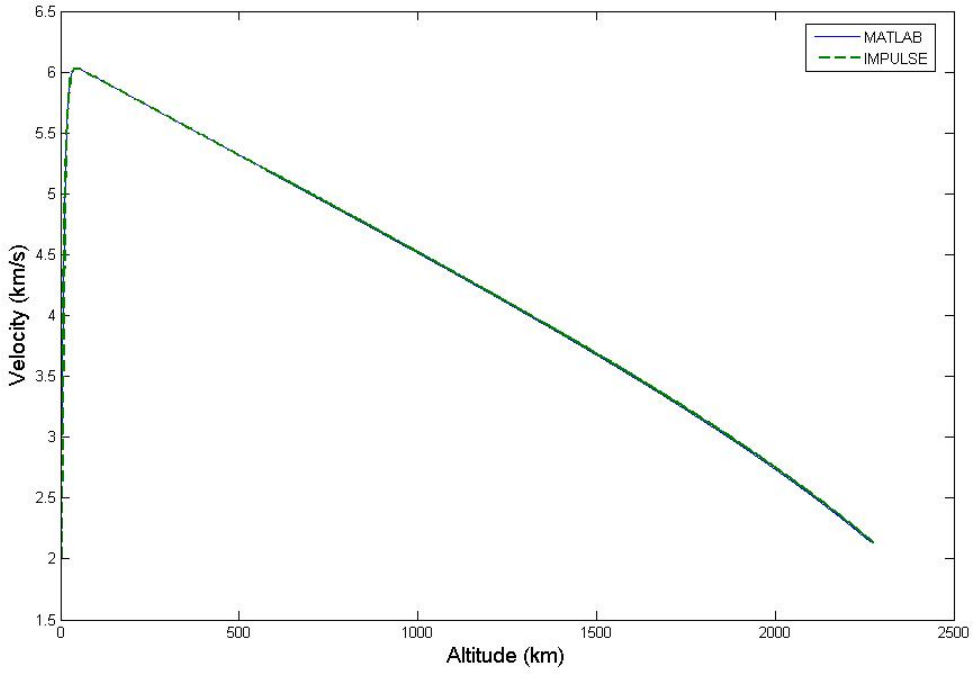

Figure 36. Velocity/Altitude Comparison for MATLAB/IMPULSE

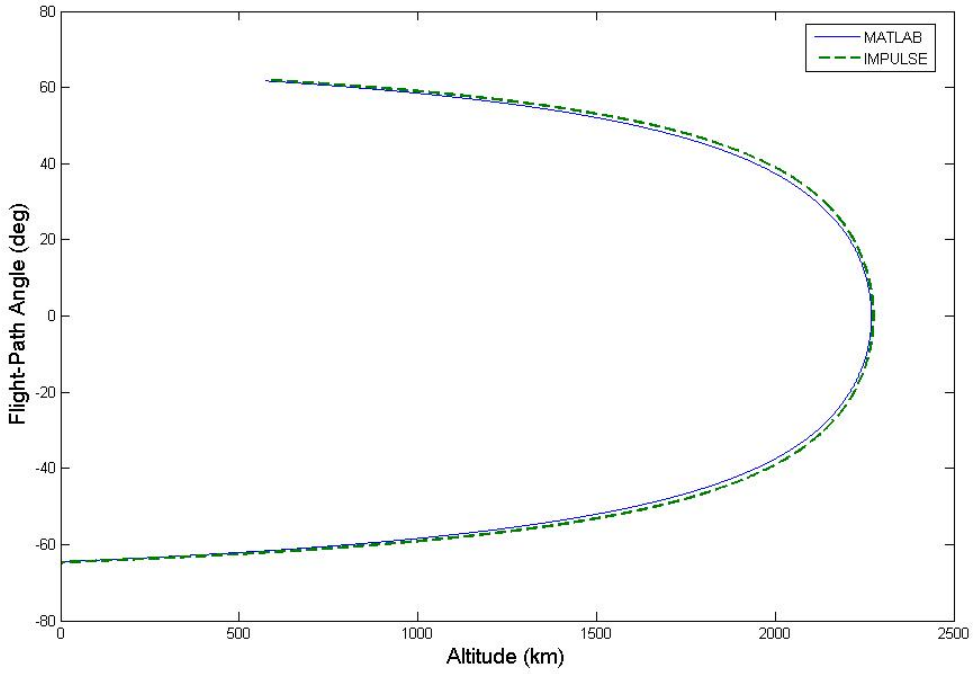

Figure 37. Flight-Path Angle/Altitude Comparison for MATLAB/IMPULSE

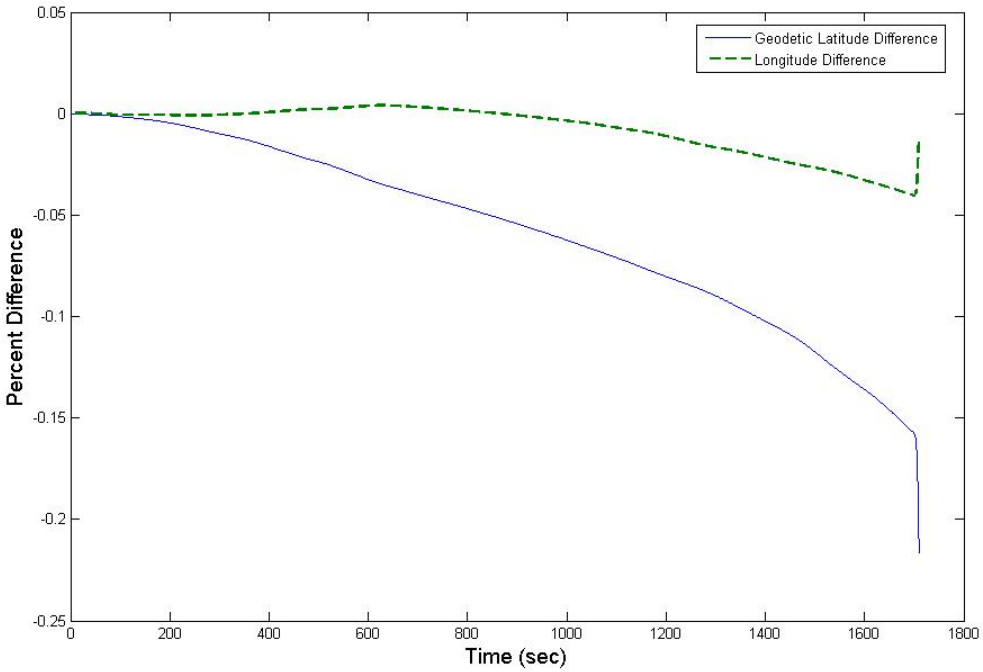

Figure 38. Comparison of Geographical Angular Data for MATLAB/IMPULSE

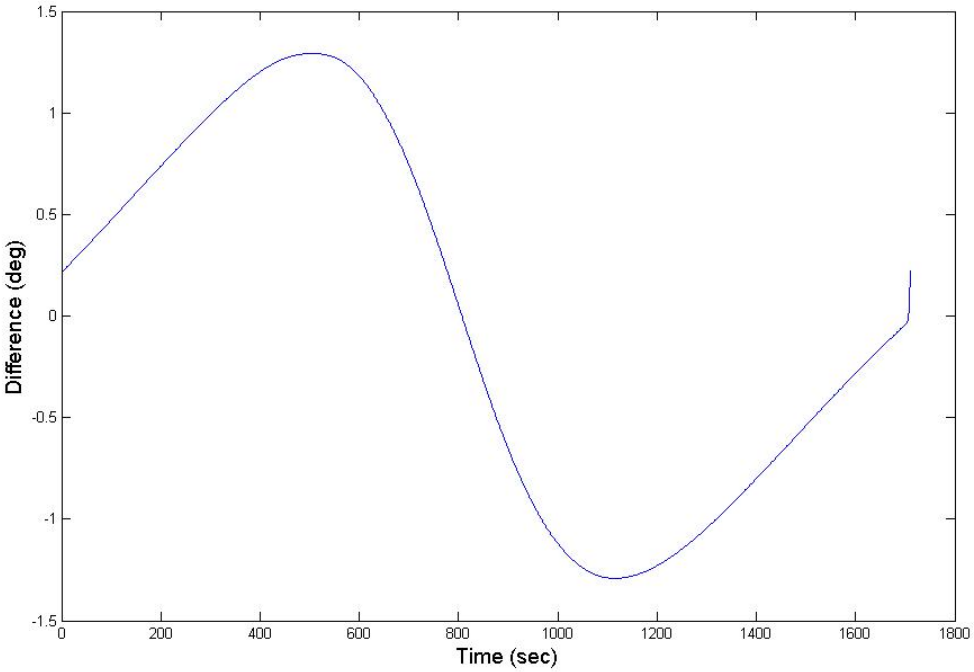

Figure 39. Comparison of Flight-Path Angles for MATLAB/IMPULSE

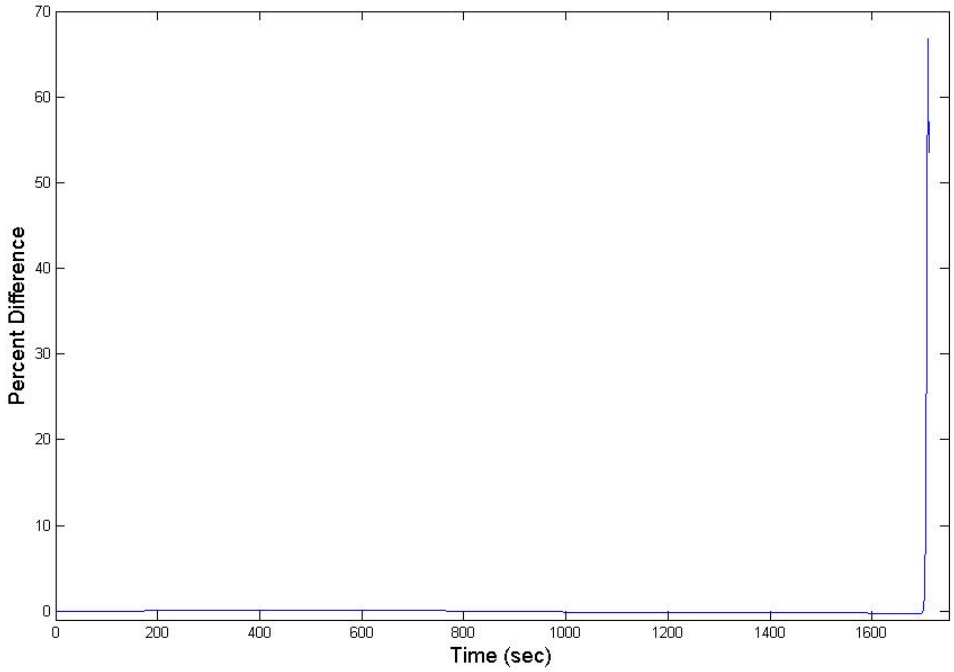

Figure 40. Comparison of Velocities for MATLAB/IMPULSE

The angular data results are essentially the same for both programs. The fact that the initial conditions matched validates the routine used in MATLAB to convert inertial coordinates to the parameters used in the equations of motion. The difference in flight-path angles was not plotted as a percent difference due to the discontinuity as the angles approached zero. The flight-path angle and altitude comparisons demonstrate the ability of the MATLAB program to simulate an orbital trajectory far from the planet's surface. Although the velocities match throughout the exoatmospheric portion of the trajectories, there is a significant deviation as the reentry vehicle penetrates the denser levels of the atmosphere. This is probably a result of the fact that IMPULSE varies C_D and C_L as a function of velocity and small angles of pitch and yaw while the MATLAB program assumes C_D is constant and used $C_L = 0$ for this simulation. Further analyis of this deviation in follow-on research may be appropriate.

Comparison with First-Order Analytic Solutions

The equations of motione described in Chapter I can be simplified and solved analytically for special cases when certain assumptions are made. Such solutions are approximations but provide a good estimate as long as the parameters fit the assumptions made for each special case.

Shallow, Gliding Entry.

The first analytic solution tested is the analysis for shallow, gliding entry. This solution assumes the vehicle produces enough lift to maintain hypersonic glide at a small flight-path angle. Once the flight-path angle exceeds the point where the small angle approximations

$$\sin\gamma \approx \gamma \tag{74}$$

$$\cos\gamma \approx 1 \tag{75}$$

no longer hold true, the analysis is no longer valid.

The solved equations of motion for shallow, gliding entry are given by (Hicks, 2005:81-86)

$$\eta = \frac{\rho S C_D}{2m\beta} \tag{76}$$

where η is a convenient way to express the altitude non-dimensionally as a function of density. The relative velocity is related to the altitude by:

$${}^{R}V^2 = \frac{g_0 r_0}{\beta r_0 \eta \left(\frac{C_L}{C_D}\right) + 1} \tag{77}$$

The flight-path angle is given by:

$$\gamma = \frac{-2g_0}{\left(\frac{C_L}{C_D}\right)\beta\, {}^{R}V^2} \tag{78}$$

Ignoring the gravity force along the flight path, the acceleration is:

$$\frac{d\,{}^{R}V}{dt} = -\beta\eta\, {}^{R}V^2 \tag{79}$$

The input parameters for the comparison with the shallow, gliding entry analytic solution are intended to simulate a large high-drag vehicle with moderate lift and are given in Table (4). The graphical comparisons are shown in Figures (36) – (40).

Table (4): Parameters for Shallow, Gliding Entry Comparison

Mass:	99000 kg
C_L:	.5
C_D:	1.0
S:	20 m^2
r(0):	6438140 m
$\gamma(0)$:	-.01304 rad
${}^{R}V(0)$:	7571.7 m/s
$\phi(0)$:	.5236 rad
$\theta(0)$:	0 rad
$\psi(0)$:	.64649 rad
σ:	0 rad

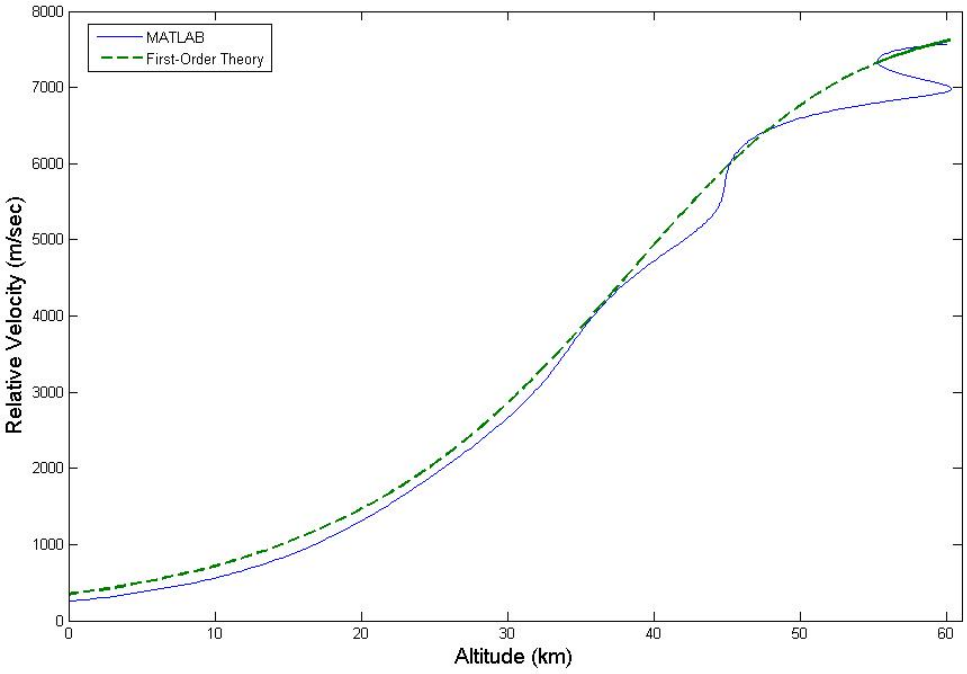

Figure 41. Velocity/Altitude Comparison for Shallow, Gliding Entry

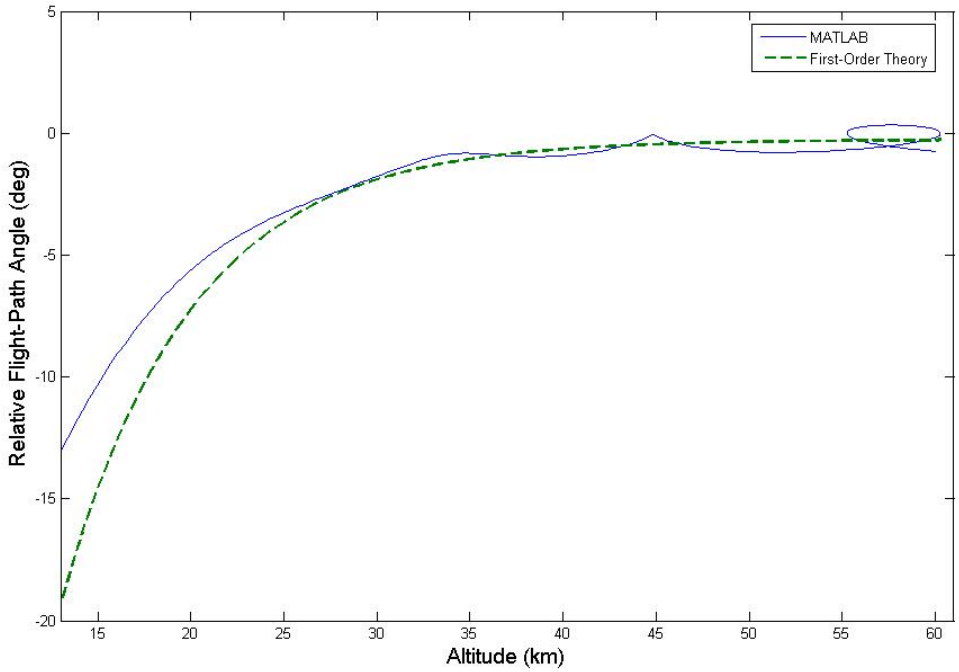

Figure 42. Flight-Path Angle/Altitude Comparison for Shallow, Gliding Entry

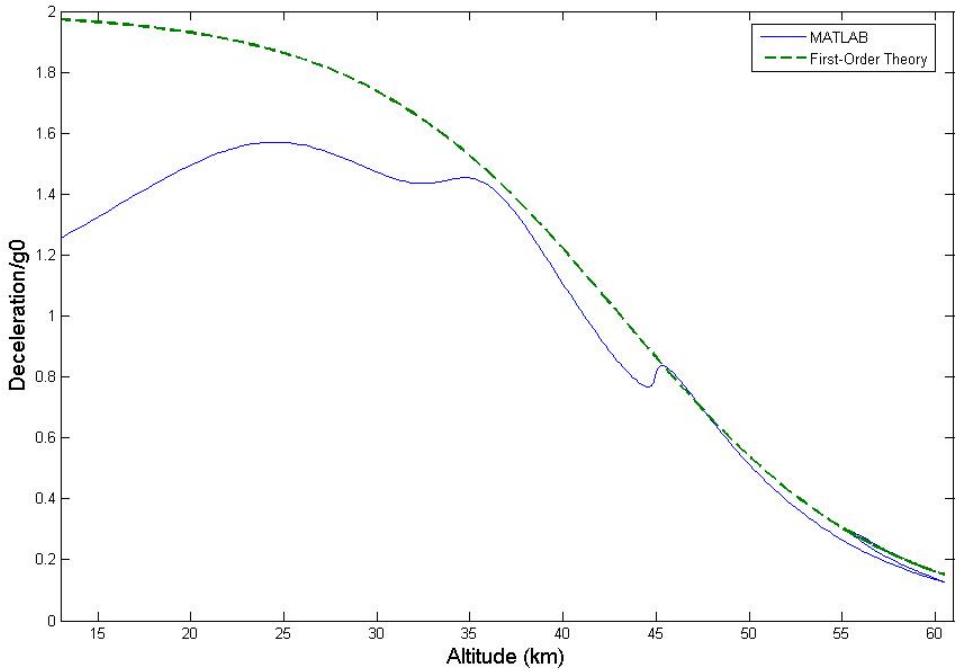

Figure 43. Deceleration/Altitude Comparison for Shallow, Gliding Entry

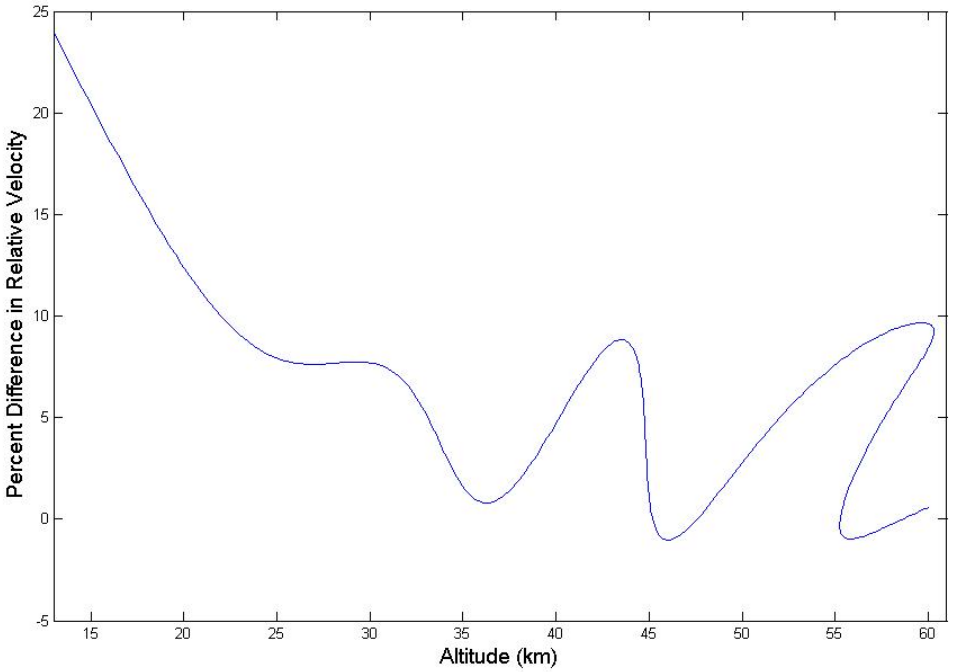

Figure 44. Comparison of Velocities for Shallow, Gliding Entry

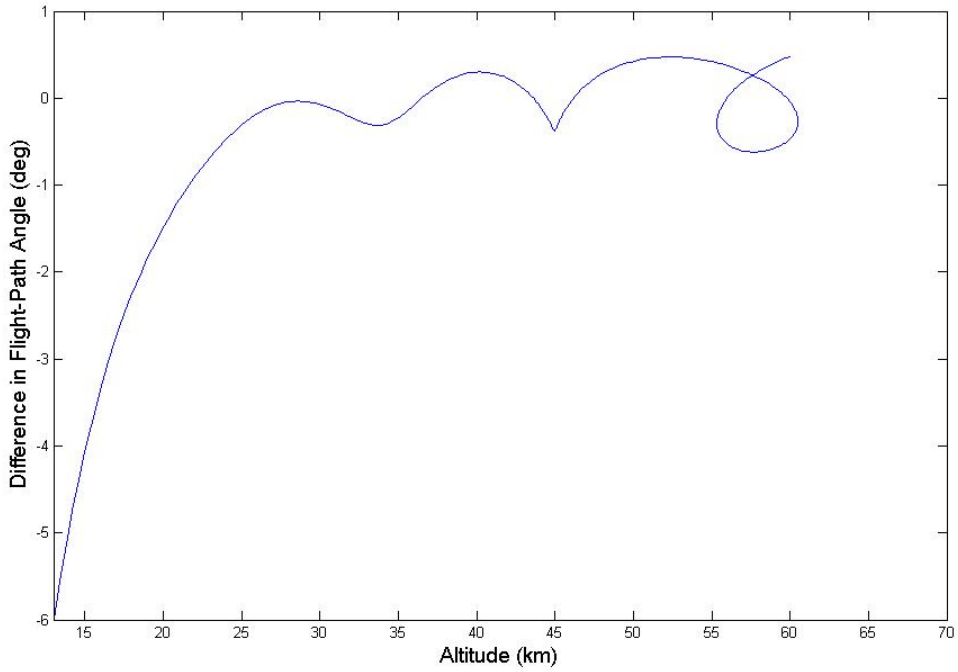

Figure 45. Comparison of Flight-Path Angles for Shallow, Gliding Entry

These results validate that the MATLAB program has been coded correctly as they match very well in the portion of the trajectory where the first-order solution is accurate. Not surprisingly, the first-order solution fails to properly predict the vehicle's behavior when the small-angle approximations for γ are no longer valid, as illustrated particularly well in Figure (37). The predicted velocity, flight-path angle, and deceleration all begin to diverge when γ decreases below about -3^0, as shown in Figures (38) – (40).

Since the deviations in the comparison with first-order theory were obviously the result of the breakdown of the small-angle approximations for γ, a quick look at this problem using a second-order solution is appropriate. Loh's unified theory for entry trajectories reveals that the relative flight-path angle is given by (Hicks, 2006:148):

$$\cos\gamma = \frac{\cos\gamma_e + \frac{C_L}{C_D}(\eta - \eta_e)}{1 + \frac{1}{\beta r_0}\left(\frac{g_0 r_0}{{}^R V^2} - 1\right)\left(1 - \frac{\eta_e}{\eta}\right)} \tag{80}$$

Figures (41) and (42) show the results, once again using the parameters shown in Table (4). The flight-path angle predicted by the second-order approximations matches the angle predicted by the MATLAB program much more closely than the first-order solution, especially as the flight-path angle decreases below -3^0. These results further bolster confidence in the MATLAB routine and help explain the deviations encountered with first-order theory as the flight-path angle steepens.

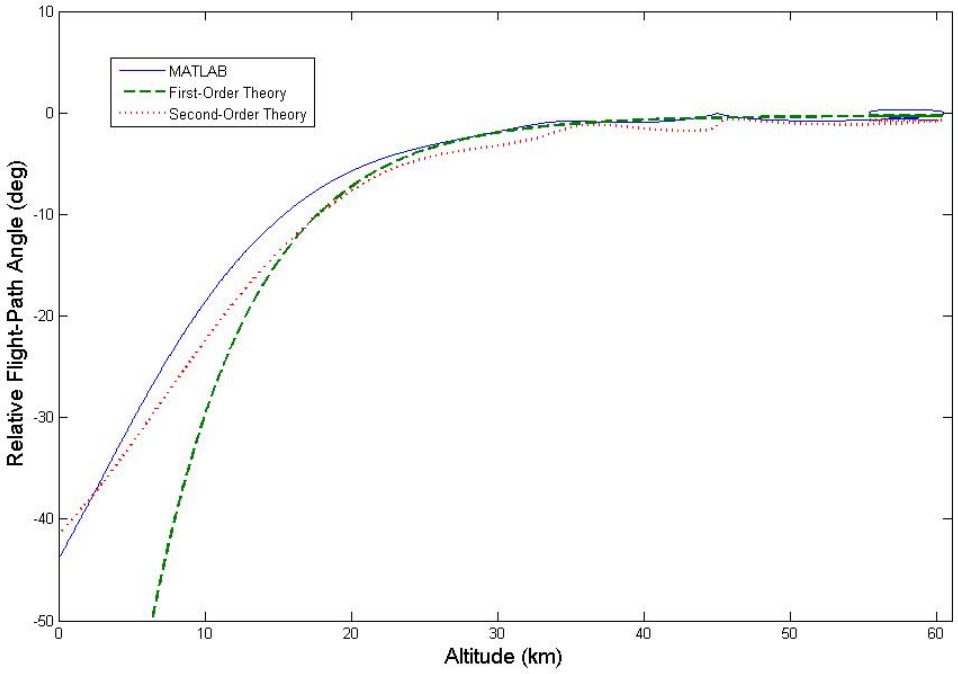

Figure 46. Second-Order Flight-Path Angle Comparison for Shallow, Gliding Entry

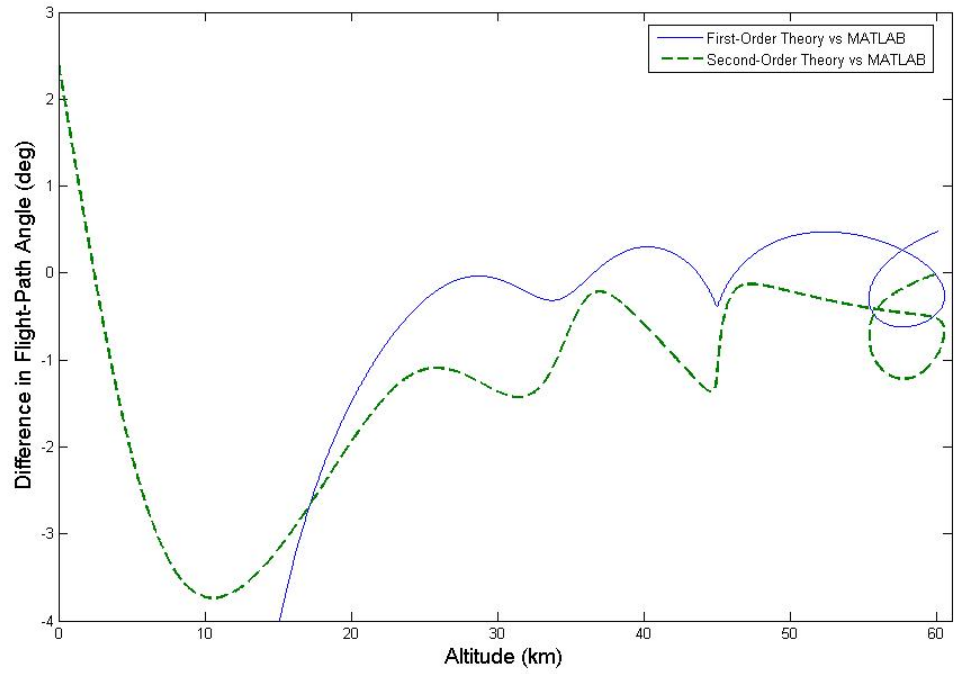

Figure 47. Second-Order Comparison of Flight-Path Angles for Shallow, Gliding Entry

Steep, Gliding Entry.

The second comparison with analytical solutions tests the case of steep entry with lift at near-circular velocity. Using the assumption that drag and lift are the dominant factors affecting the trajectory, Hicks derives the approximate solutions for relative velocity and relative flight-path angle and deceleration (a_{decel}) as (2005:55-57):

$$\cos\gamma = \gamma_e + \frac{C_L}{C_D}\left(\eta - \eta_e\right) \tag{81}$$

$${}^{R}V^2 = {}^{R}V_e^2 \exp\left(\frac{2\gamma_e - \gamma}{\left(\frac{C_L}{C_D}\right)}\right) \tag{82}$$

$$a_{decel} = \beta\, {}^{R}V^2 \left(\frac{\cos\gamma - \cos\gamma_e}{\left(\frac{C_L}{C_D}\right)} + \eta_e\right) \exp\left(\frac{2\left(\gamma - \gamma_e\right)}{\left(\frac{C_L}{C_D}\right)}\right) \tag{83}$$

The subscript *e* denotes the initial condition at entry.

The model used for this test is a small, dense, streamlined vehicle with lift entering at a near-vertical flight-path angle. The initial parameters are listed in Table (5) and the results are shown in Figures (43) – (47).

Table (5): Parameters for Steep, Gliding Entry Comparison

Mass:	50 kg
C_L:	.03
C_D:	.01
S:	.01 m^2
r(0):	6478140 m
$\gamma(0)$:	-85^0
${}^R V(0)$:	5372 m/s
$\phi(0)$:	.600588 rad
$\theta(0)$:	-1.8075 rad
$\psi(0)$:	1.916 rad
σ:	0 rad

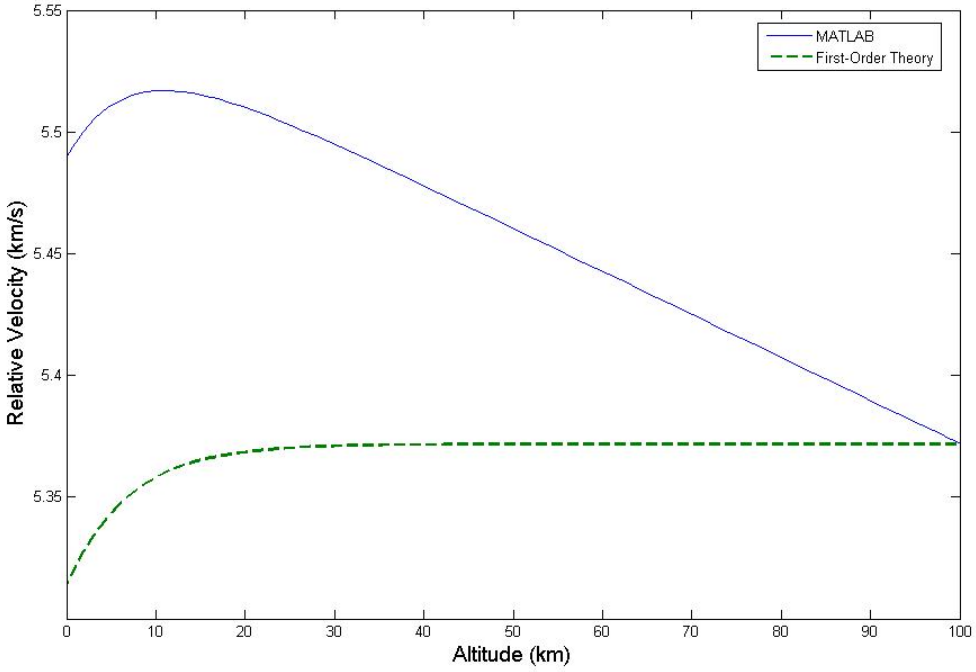

Figure 48. Velocity/Altitude Comparison for Steep, Gliding Entry

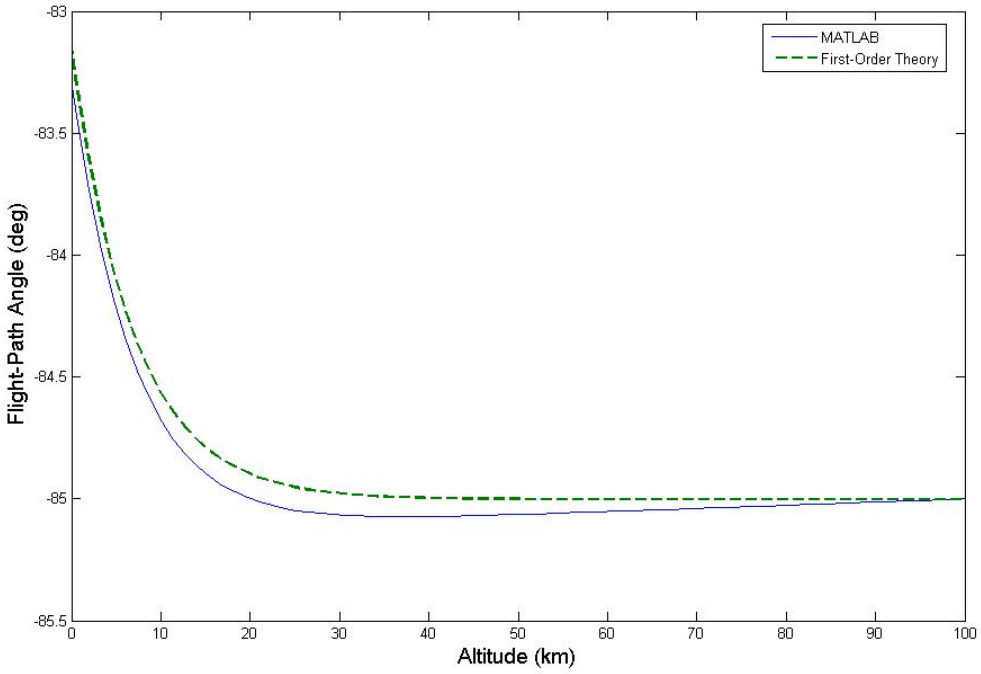

Figure 49. Flight-Path Angle/Altitude Comparison for Steep, Gliding Entry

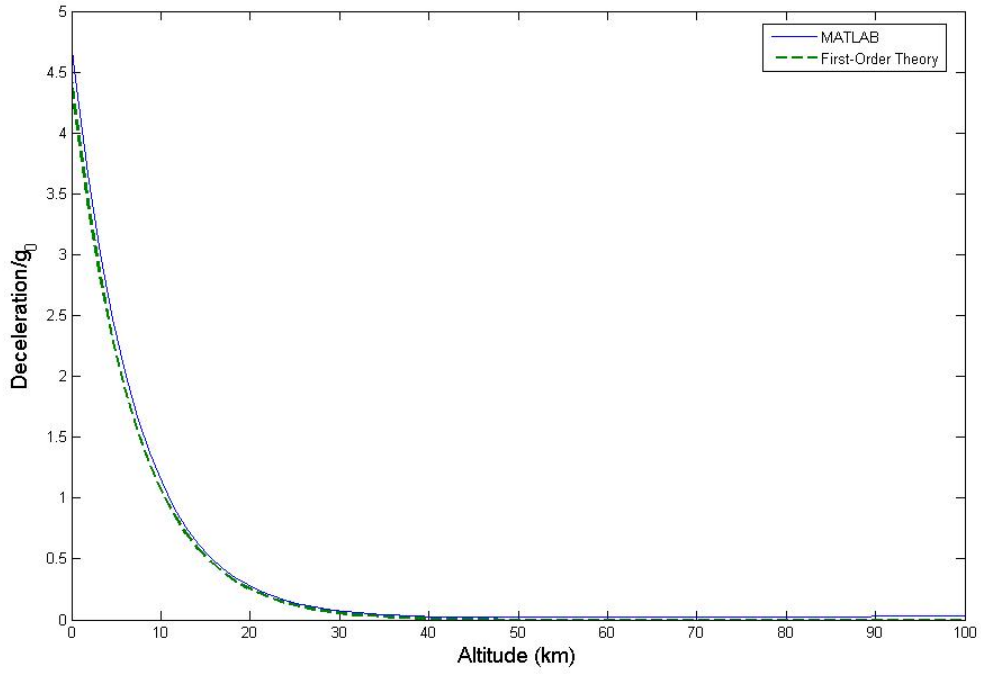

Figure 50. Deceleration/Altitude Comparison for Steep, Gliding Entry

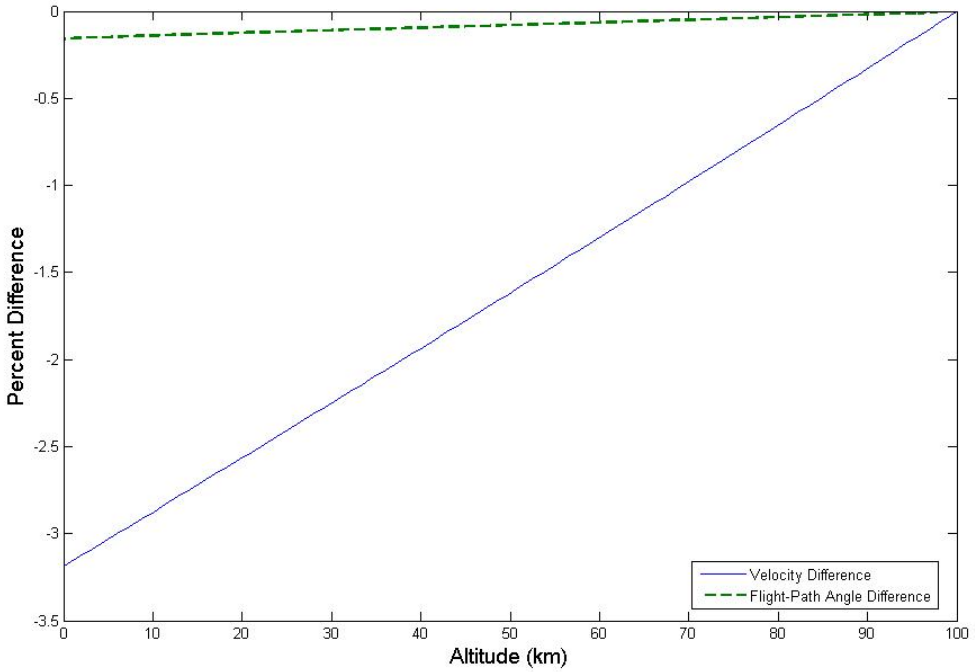

Figure 51. Comparison of Data for Steep, Gliding Entry

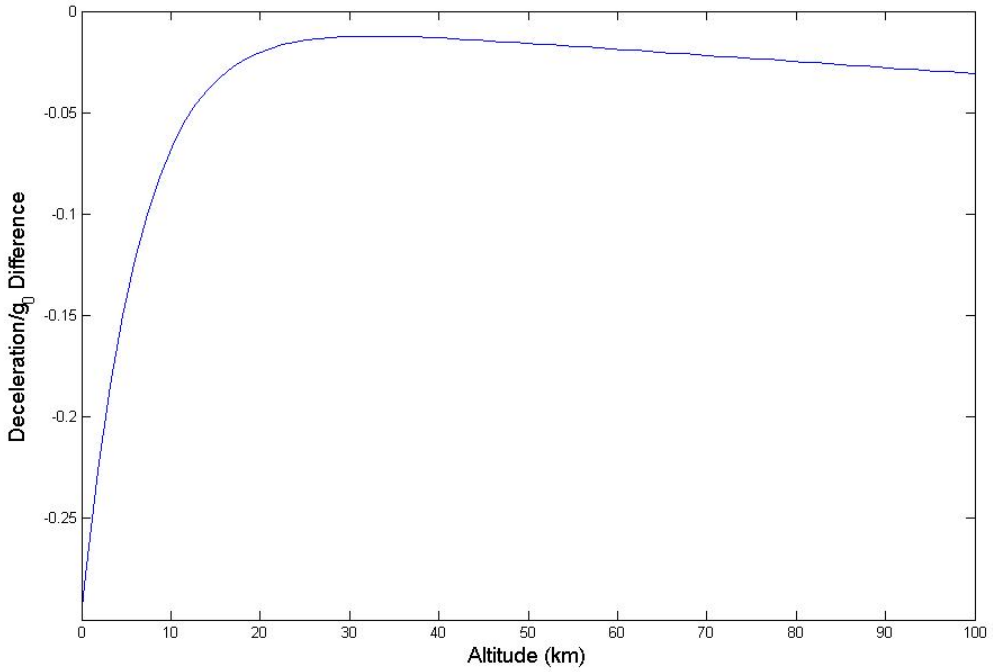

Figure 52. Comparison of Deceleration difference for Steep, Gliding Entry

The close agreement of these results strongly suggest that the equations have been coded correctly in MATLAB and that the program produces reliable trajectories for steep, gliding entry problems.

Steep, Ballistic Entry.

First-Order approximation for steep, ballistic entry models a hypersonic vehicle with no lift at a near-vertical flight-path angle. The solution makes the assumption that the flight-path angle remains constant throughout the trajectory. The simplified equations of motion for relative velocity and relative flight-path angle are derived by Hicks (2006:118-121).

Since the flight-path angle is a constant:

$$\gamma = \gamma_e \tag{84}$$

For simplification the variable α is defined as

$$\alpha = \frac{-2\eta}{\sin\gamma_e} \tag{85}$$

and a constant A defined as

$$A = \frac{{}^{R}V_e^2 \beta \exp(\alpha_e)}{2g_0} - Ei(\alpha_e) \tag{86}$$

where $Ei(\alpha_e)$ is the exponential integral α_e. The relative velocity is defined as:

$${}^{R}V^2 = \frac{2g_0 \exp(-\alpha)\left(A + Ei(\alpha)\right)}{\beta} \tag{87}$$

Once again, the subscript *e* denotes the initial entry condition.

The initial parameters for the comparison between the MATLAB program and first-order analytical solution for steep, ballistic entry are shown in Table (6) and the results are shown graphically in Figures (48) – (50).

Table (6): Parameters for Steep, Ballistic Entry Comparison

Mass:	13 kg
C_L:	0
C_D:	.08575
S:	.036609615 m^2
r(0):	6778140 m
$\gamma(0)$:	-85^0
${}^R V(0)$:	10193 m/s
$\phi(0)$:	.58059 rad
$\theta(0)$:	-1.8875 rad
$\psi(0)$:	1.7 rad
σ:	0 rad

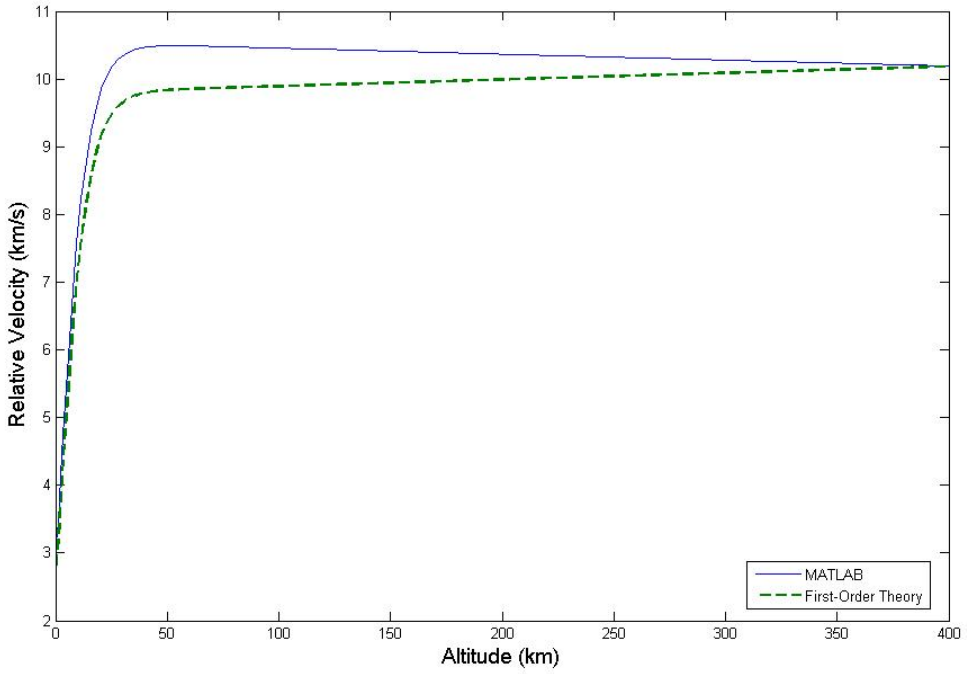

Figure 53. Velocity/Altitude Comparison for Steep, Ballistic Entry

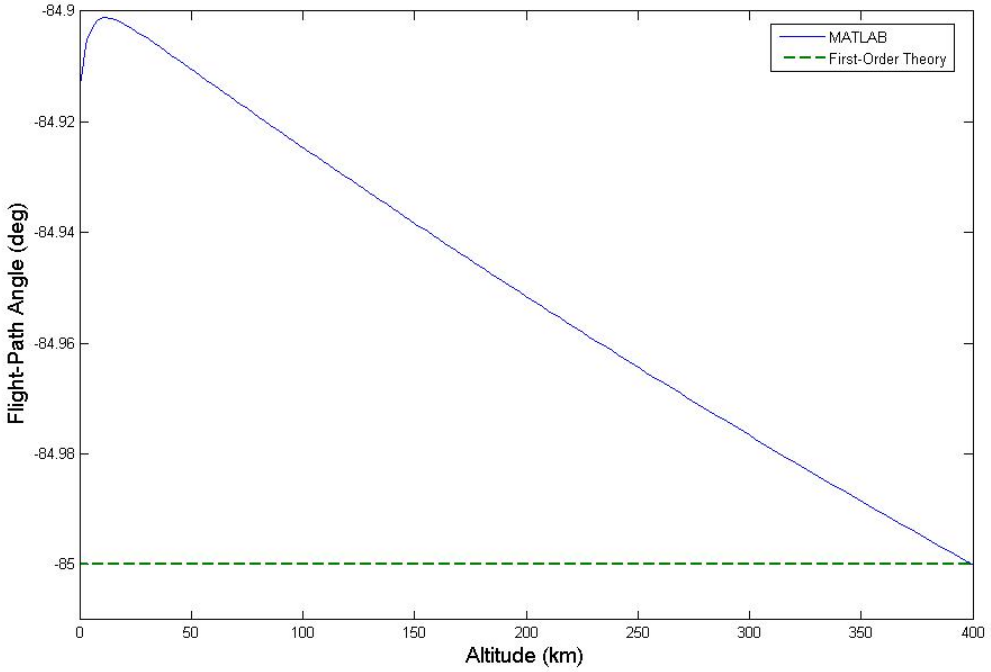

Figure 54. Flight-Path Angle/Altitude Comparison for Steep, Ballistic Entry

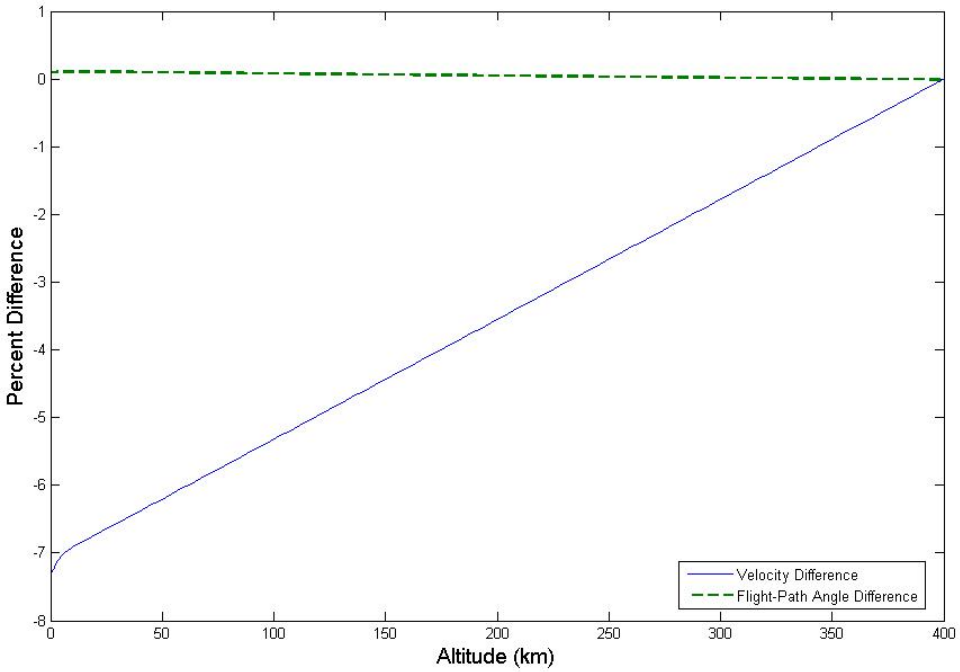

Figure 55. Comparison of Data for Steep, Ballistic Entry

Once again, the first-order theory matches well with the MATLAB program. The small deviation in flight-path angle is expected because the analytical solution treats this parameter as a constant, as expressed in Equation (84). The velocity difference is consistent with the previous comparisons with first-order solutions.

Medium, Gliding Entry at Supercircular Velocity.

The final test is another look at medium, gliding entry, but this time there is no restriction that the initial velocity must be at near-circular speed. Hicks shows how Wang and Ting developed their approximate solution for this condition with the assumption that the flight-path angle remains relatively small (2006:104-107). Their simplified equations for flight-path angle and relative velocity are as follows:

$$\gamma = -\left[\gamma_e^2 - 2\frac{C_L}{C_D}(\eta - \eta_e) - \frac{2}{\beta r_0}\left(1 - \frac{g_0 r_0}{{}^R V_e^2}\right)\ln\left(\frac{\eta}{\eta}\right)\right]^{\frac{1}{2}} \tag{88}$$

$${}^R V^2 = V_e^2 \exp\left(\frac{2(\gamma_e - \gamma)}{\frac{C_L}{C_D} + \frac{1}{\beta r_0 \eta}\left(1 - \frac{g_0 r_0}{{}^R V_e^2}\right)}\right) \tag{89}$$

As with shallow, gliding entry at near-circular speeds, the acceleration remains

$$\frac{d\,{}^R V}{dt} = -\beta\eta\,{}^R V^2 \tag{90}$$

The simulated model for the comparison with the first-order solution for medium, gliding entry at supercircular velocity is a small lifting body entering the atmosphere at a moderate flight-path angle with high velocity. The initial parameters are listed in Table (7) and the results are shown in Figures (51) – (54).

Table (7): Parameters for Medium, Gliding Entry at Supercircular Velocity Comparison

Mass:	2000 kg
C_L:	.075
C_D:	.3
S:	20 m^2
r(0):	6478140 m
$\gamma(0)$:	-35^0
${}^R V(0)$:	9847 m/s
$\phi(0)$:	.5236 rad
$\theta(0)$:	0 rad
$\psi(0)$:	1.146 rad
σ:	0 rad

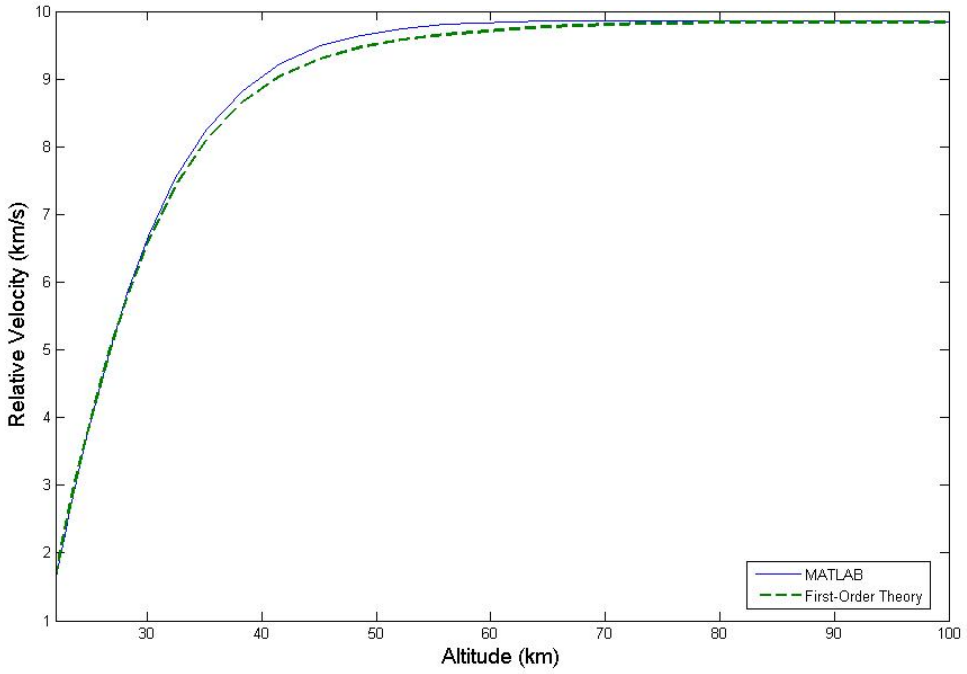

Figure 56. Velocity/Altitude Comparison for Medium, Gliding Entry at Supercircular Speed

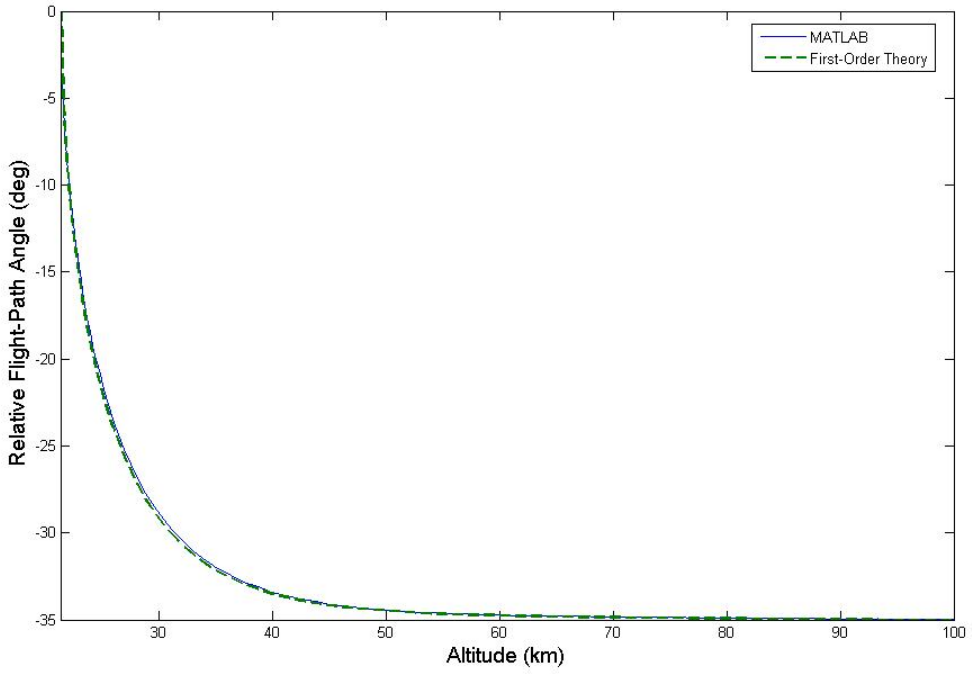

Figure 57. Flight-Path Angle/Altitude Comparison for Medium, Gliding Entry at Supercircular Speed

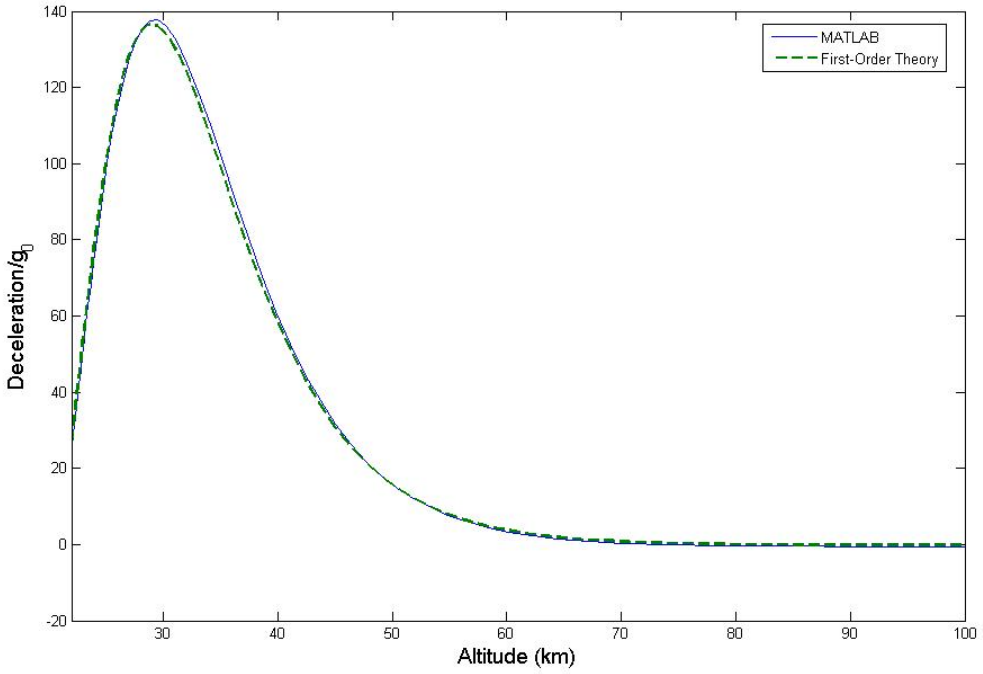

Figure 58. Deceleration/Altitude Comparison for Medium, Gliding Entry at Supercircular Speed

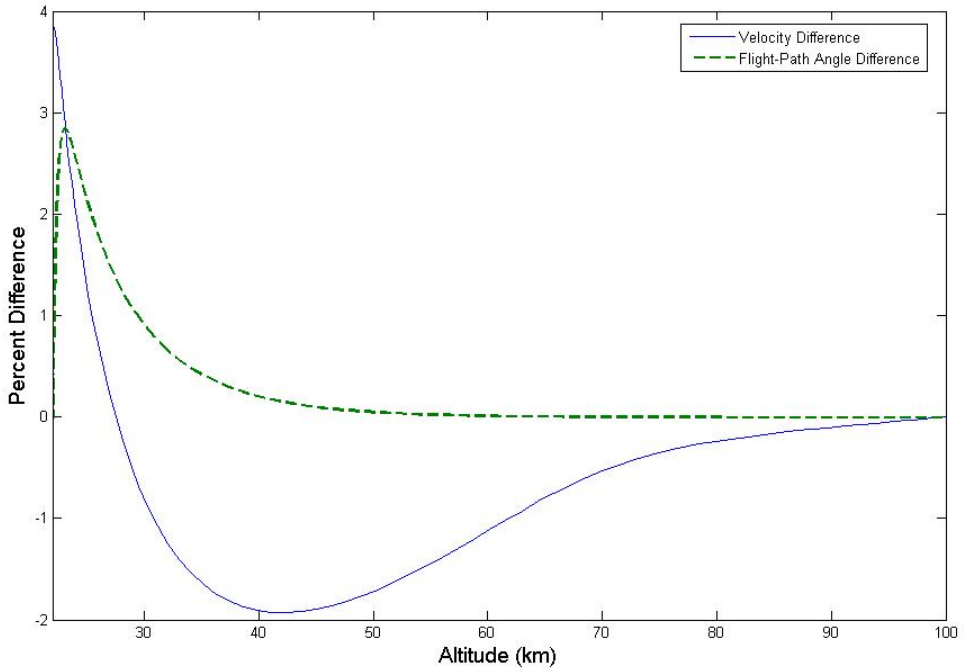

Figure 59. Comparison of Data for Medium, Gliding Entry at Supercircular Speed

Due to the fact that the first-order solution for flight-path angle becomes imaginary when the quantity in brackets on the right side of Equation (88) goes below zero, the comparison could only be plotted to an altitude of 21.3 km. Below this altitude the first-order solutions for flight-path angle, relative velocity, and acceleration became complex numbers and were no longer valid for comparison purposes. Up to the point where the analytic solution fell apart, the MATLAB program agreed closely with the first-order approximations, further solidifying the fact that the program is operating properly.

VI. Conclusion and Recommendations

Conclusion

For this research an algorithm was developed using MATLAB to model a trajectory for a vehicle entering the Earth's atmosphere. The algorithm used the equations of motion developed in Chapter II and established the initial conditions using the methods shown in Chapter III to establish the simulation. The program was then compared to previously established software that were also designed to predict entry trajectories. It was also compared to first-order approximations for specific types of reentry. The program code is listed in the Appendix.

The MATLAB code generated fulfills the goal of this research of creating a program to quickly generate atmospheric entry simulations. It compares very favorably with TARGET, a FORTRAN program with similar objectives, and IMPULSE, another MATLAB program focused on simulating ICBMs. In both cases, the methods used in Chapter III to establish the initial conditions yielded perfect results, confirming that the equations used were valid and coded correctly.

Comparison with the first-order approximations further solidified the validity of the MATLAB algorithm. The MATLAB program routinely produced results that closely matched the first-order solutions over those portions of the trajectory one would expect the analytic approximations to maintain accuracy.

Although the model used in this research is not all encompassing, it provides the capability to produce reliable results for systems-level studies. The program was intended to be simple and quick for initial analysis and to utilize MATLAB due to the

language's emerging widespread employment. The algorithm is not intended to be a replacement for more comprehensive routines that already exist, such as POST.

Recommendations for Future Work

The MATLAB routine developed in this research lends itself to future development. There are several areas of the algorithm outside the scope of this research that could be improved. The software is quite adaptable and implementing future adaptations should not be too difficult.

This research used a simple exponential model as an adequate approximation of the Earth's atmosphere. Improvement could be made by more completely modeling the atmosphere as several layers, each with a different scale height.

For simplification, this research made the assumption that C_L and C_D are constant throughout the atmospheric entry. In reality, both coefficients are functions of angle of attack and velocity. An algorithm could be implemented to allow changes in angle of attack at given altitudes or velocities and to calculate the corresponding coefficients of lift and drag.

Just as angle of attack was held constant, the bank angle (σ) was also assumed to be constant during entry. By allowing variations in σ and angle of attack, a future development of the software would be able to simulate various trajectories that use lift to maneuver the vehicle during reentry.

This research also assumed that the reentry vehicle had no propulsion system. The equations of motion as shown by Hicks (2006:58) account for a thrust component. These equations could be substituted for the ones shown in Chapter II and adapted to

simulate a vehicle that uses thrust or a combination of thrust and lift to maneuver during reentry.

Finally, future research may be interested in the amount of aerodynamic heating expected during reentry. Vinh (1980:141-156) shows how heating rates vary with velocity and atmospheric density. These analytic solutions could be added to the existing code to generate the heating information for the entire trajectory.

In all of these cases, the MATLAB program generated in this research would be the core program. The routine would maintain the ability to generate quick and simple results with a reasonably high degree of accuracy for high-level studies.

Appendix: MATLAB Code

Core Code

This is the MATLAB core code for simulating entry trajectories (*reentry.m*). It uses the algorithms developed in this research. The codes for operating the graphical user interfaces (*inputs.m, vpoint.m, eci.m* and *orb.m*) and the ordinary differential equation solvers (*spherical.m, ellipsoid.m*) are listed following the core code.

```
% Capt Robert Jameson
% AFIT/GSS/ENY/06-08
% 14 January 2006
% This program simulates the reentry trajectory of a vehicle with perfect
%   knowledge of the vehicle parameters and the initial conditions for the
%   equations of motion.

clc

% Clear acceleration data
clear 'A'

% Surface density of atmosphere in kg/m^3 at STP (assumption)
rhos=1.54793;

% Average scale height in meters (assumption)
H=6935;

% Calculate inverse of scale height (beta)
beta=1/H;

% Radius of the Earth at the equator in meters (WGS-84 value)
re=6378137;

% Radius of the Earth at the poles in meters (WGS-84 value)
rp=6356752.3142;

% This equation calculates the eccentricity of the Earth's atmosphere (ee)
ee=1-rp/re;

% WGS-84 acceleration due to gravity at the radius of equator (m/s^2)
```

```
g0=9.798;

% WGS-84 rotation rate of Earth (rad/s)
omegae=.000072921151467;

% Create initial user inputs if none exist
 if ~exist('user_inputs')

% Run GUI (inputs.m) to get user inputs for the shape of the Earth's
%   atmosphere and the desired coordinate system
   inputs
   uiwait(inputs)

% Save values for coordinate system (1=vehicle pointing, 2=ECI,
%   3=classic orbital elements) and shape of atmosphere (1=spherical,
%   2=ellipsoid) into .mat files
   load('var')
   load('shape')

% If the vehicle pointing system is selected, run GUI for user inputs for
%   vehicle parameters and initial conditions (vpoint.m)
   if var==1
      vpoint
      uiwait(vpoint)

% Save values for vehicle parameters (m=mass, Cl=coefficient of lift,
%   Cd=coefficient of drag, S=reference surface area) and initial
%   conditions (ri=radius, gammar=relative flight-path angle,
%   vr=relative velocity, thetai=longitude, phii=geocentric latitude,
%   psir=relative heading, sigma=bank angle) into .mat files
      load('m')
      load('Cl')
      load('Cd')
      load('S')
      load('ri')
      load('gammar')
      load('vr')
      load('thetai')
      load('phii')
      load('psir')
      load('sigma')
   end

% If the ECI system is selected, run GUI for user inputs for
%   vehicle parameters and initial conditions (eci.m)
   if var==2
```

```
        eci
        uiwait(eci)

% Save values for vehicle parameters (m=mass, Cl=coefficient of lift,
%   Cd=coefficient of drag, S=reference surface area) and initial
%   conditions (x,y,z=inertial position, vx,vy,vz=inerial velocity,
%   sigma=bank angle, hour,min,sec=Sidereal time) into .mat files
        load('m')
        load('Cl')
        load('Cd')
        load('S')
        load('x')
        load('y')
        load('z')
        load('vx')
        load('vy')
        load('vz')
        load('sigma')
        load('hour')
        load('min')
        load('sec')
    end

% If the classic orbital elements are selected, run GUI for user inputs for
%   vehicle parameters and initial conditions (orb.m)
    if var==3
        orb
        uiwait(orb)

% Save values for vehicle parameters (m=mass, Cl=coefficient of lift,
%   Cd=coefficient of drag, S=reference surface area) and initial
%   conditions (ri=radius, gammar=relative flight-path angle,
%   vr=relative velocity, thetai=longitude, phii=geocentric latitude,
%   psir=relative heading, sigma=bank angle) into .mat files
        load('m')
        load('Cl')
        load('Cd')
        load('S')
        load('ri')
        load('gammar')
        load('vr')
        load('thetai')
        load('phii')
        load('psir')
        load('sigma')
    end
```

```
% If the ECI system is selected, run GUI for user inputs for
%   vehicle parameters and initial conditions (eci.m)
   if var==2
      eci
       uiwait(eci)

% Save values for vehicle parameters (m=mass, Cl=coefficient of lift,
%   Cd=coefficient of drag, S=reference surface area) and initial
%   conditions (x,y,z=inertial position, vx,vy,vz=inerial velocity,
%   sigma=bank angle, hour,min,sec=Sidereal time) into .mat files
       load('m')
       load('Cl')
       load('Cd')
       load('S')
       load('x')
       load('y')
       load('z')
       load('vx')
       load('vy')
       load('vz')
       load('sigma')
       load('hour')
       load('min')
       load('sec')
   end

% If the classic orbital elements are selected, run GUI for user inputs for
%   vehicle parameters and initial conditions (orb.m)
   if var==3
       orb
       uiwait(orb)

% Save values for vehicle parameters (m=mass, Cl=coefficient of lift,
%   Cd=coefficient of drag, S=reference surface area) and initial
%   conditions (a=semi-major axis, nu=true anomaly, inc=inclination,
%   e=eccentricity, omega=argument of perigee, node=right ascension of
%   ascending node, sigma=bank angle, hour,min,sec=Sidereal time) into .mat
%   files
       load('m')
       load('Cl')
       load('Cd')
       load('S')
       load('a')
       load('nu')
       load('inc')
```

```
        load('e')
        load('omega')
        load('node')
        load('sigma')
        load('hour')
        load('min')
        load('sec')
    end

% set up user_inputs and parameters matrix if vehicle pointing system
%   selected
    if var==1
        user_inputs=[S;Cl;Cd;m;vr;gammar;sigma;thetai;phii;ri;psir];

        parameters={'Surface Area';'Cl';'Cd';'Mass';'Entry Velocity';...
        'Initial Flight-Path Angle';...
        'Bank Angle';'Longitude';'Latitude';'Initial Radius';'Heading'};

% Display user_inputs and parameters in array editor for user reference
%   and modification
        openvar('parameters');
        openvar('user_inputs');

    end

% set up user_inputs and parameters matrix if ECI system selected
   if var==2
        user_inputs=[S;Cl;Cd;m;x;y;z;vx;vy;vz;sigma;hour;min;sec];

        parameters={'Surface Area';'Cl';'Cd';'Mass';'X';'Y';...
        'Z';'X dot';'Y dot';'Z dot';'Bank Angle';'Hour';'Minute';'Second'};

% Display user_inputs and parameters in array editor for user reference
%   and modification
        openvar('parameters');
        openvar('user_inputs');

   end

% set up user_inputs and parameters matrix if classic orbital elements
%   selected
    if var==3
        user_inputs=[S;Cl;Cd;m;a;e;inc;node;omega;nu;sigma;hour;min;sec];

        parameters={'Surface Area';'Cl';'Cd';'Mass';'Semi-Major Axis';'Eccentricity';...
```

```
          'Inclination';'Node';'Perigee';'True Aanomaly';'Bank
         Angle';'Hour';'Minute';'Second'};

% Display user_inputs and parameters in array editor for user reference
%   and modification
        openvar('parameters');
        openvar('user_inputs');

   end

 end

%  Allow editing of user_inputs in display editor
if var==1
    S=user_inputs(1);
    Cl=user_inputs(2);
    Cd=user_inputs(3);
    m=user_inputs(4);
    vr=user_inputs(5);
    gammar=user_inputs(6);
    sigma=user_inputs(7);
    thetai=user_inputs(8);
    phii=user_inputs(9);
    ri=user_inputs(10);
    psir=user_inputs(11);

 end


if var==2

%  Allow editing of user_inputs in display editor
    S=user_inputs(1);
    Cl=user_inputs(2);
    Cd=user_inputs(3);
    m=user_inputs(4);
    x=user_inputs(5);
    y=user_inputs(6);
    z=user_inputs(7);
    vx=user_inputs(8);
    vy=user_inputs(9);
    vz=user_inputs(10);
    sigma=user_inputs(11);
    hour=user_inputs(12);
    min=user_inputs(13);
    sec=user_inputs(14);
```

```
% Develop initial conditions when ECI system is selected
% Calculate initial radius from inertial position (ri)
   ri=(x^2+y^2+z^2)^.5;

% Calculate magnitude of inertial velocity (vi)
   vi=(vx^2+vy^2+vz^2)^.5;

% Generate vectors for inertial position and velocity
   Rxyz=[x;y;z];
   Vxyz=[vx;vy;vz];

% Calculate angular momentum vector (H)
   H=cross(Rxyz,Vxyz);

% Calculate magnitude of angular momentum (h)
   h=(H(1)^2+H(2)^2+H(3)^2)^.5;

% Calculate the inertial flight-path angle (gammai) and find the
% correct sign.
   if dot(Rxyz,Vxyz)>0

      gammai=acos(h/(ri*vi));
   else
      gammai=-acos(h/(ri*vi));
   end

% Calculate the geocentric latitude (phii)
   phii=asin(z/ri);

% Calculate the longitude assuming t=0 (theta)
   theta=asin(y/(x^2+y^2)^.5);

% Correct longitude for sign ambiguity
   if x<0
      theta=pi-theta;
   end

% Correct longitude for Greenwich sidereal time (thetai)
  thetai=theta-(user_inputs(12,1)*3600+user_inputs(13,1)*60+...
     user_inputs(14,1))*omegae;

% Ensure the magnitude of thetai is not greater than 2pi
  if thetai<-2*pi
     thetai=thetai+2*pi
  end
```

```
% Calculate the inclination of the orbit (inc)
   inc=acos(H(3)/h);

 end

% Develop initial conditions when orbital elements are selected
 if var==3

% Allow editing of user_inputs in display editor
     S=user_inputs(1);
     Cl=user_inputs(2);
     Cd=user_inputs(3);
     m=user_inputs(4);
     a=user_inputs(5);
     e=user_inputs(6);
     inc=user_inputs(7);
     node=user_inputs(8);
     omega=user_inputs(9);
     nu=user_inputs(10);
     sigma=user_inputs(11);
     hour=user_inputs(12);
     min=user_inputs(13);
     sec=user_inputs(14);

% Earth's gravitational parameter (m^3/s^2)
   mu=3.98601*10^14;

% Calculate the semi-latus rectum of the orbit
   p=a*(1-e^2);

% Calculate radius (ri)
   ri=a*(1-e^2)/(1+e*cos(nu));

% Calculate P-component of velocity (vp) in the perifocal coordinate system
   vp=(mu/p)^.5*(-sin(nu));

% Calculate Q-component of velocity (vq) in perifocal system
   vq=(mu/p)^.5*(e+cos(nu));

% Establish velocity vector in perifocal system
   V=[vp;vq];

% Establish radius vector in perifocal system
   R=[ri*cos(nu);ri*sin(nu)];
```

```
% Calculate position in ECI system from perifocal system
   x=R(1)*(cos(node)*cos(omega)-sin(node)*sin(omega)*cos(inc))+R(2)*...
      (-cos(node)*sin(omega)-sin(node)*cos(omega)*cos(inc));

   y=R(1)*(sin(node)*cos(omega)+cos(node)*sin(omega)*cos(inc))+R(2)*...
      (-sin(node)*sin(omega)+cos(node)*cos(omega)*cos(inc));

   z=R(1)*(sin(omega)*sin(inc))+R(2)*(cos(omega)*sin(inc));

% Calculate velocity in ECI system from perifocal system
   vx=V(1)*(cos(node)*cos(omega)-sin(node)*sin(omega)*cos(inc))+V(2)*...
      (-cos(node)*sin(omega)-sin(node)*cos(omega)*cos(inc));

   vy=V(1)*(sin(node)*cos(omega)+cos(node)*sin(omega)*cos(inc))+V(2)*...
      (-sin(node)*sin(omega)+cos(node)*cos(omega)*cos(inc));

   vz=V(1)*(sin(omega)*sin(inc))+V(2)*(cos(omega)*sin(inc));

   vi=(vx^2+vy^2+vz^2)^.5;

% Establish radius and velocity vectors in ECI system
   Rxyz=[x;y;z];
   Vxyz=[vx;vy;vz];

% Calculate angular momentum vector
   H=cross(Rxyz,Vxyz);

% Calculate magnitude of angular momentum
   h=(H(1)^2+H(2)^2+H(3)^2)^.5;

% Calculate inertial flight-path angle (gammai) and correct for sign
%   ambiguity.
   if nu>pi
      gammai=-acos(h/(ri*vi));
   else
      gammai=acos(h/(ri*vi));
   end

% Calculate geocentric latitude
   phii=asin(z/ri);

% Calculate the longitude assuming t=0 (theta)
   theta=asin(y/(x^2+y^2)^.5);

% Correct for ambiguity of theta
   if x<0
```

```
        theta=pi-theta;
    end
% Correct longitude for Greenwich sidereal time (thetai)
   thetai=theta-(user_inputs(12,1)*3600+user_inputs(13,1)*60+...
       user_inputs(14,1))*omegae;

% Ensure the magnitude of thetai is not greater than 2pi
   if thetai<-2*pi
       thetai=thetai+2*pi;
   end

 end

% Continue determing initial conditions when ECI system or classic orbital
%   elements are selected
if var>1

% Calculate the local horizontal component (vh) of the inertial velocity
    vh=vi*cos(gammai);

% Calculate the local velocity of the Earth's rotation (ve)
    ve=ri*omegae*cos(phii);

% Find the right ascension of the ascending node (node) when the ECI
%   system is selected
    if var==2
    kH=cross([0;0;1],H);
    N=kH/((kH(1)^2+kH(2)^2+kH(3)^2)^.5);
    node=acos(N(1));

% Correct for sign ambiguity of right ascension of ascending node
         if N(2)<0
            node=-node;
         end

    end

% Calculate the inertial heading (psii)
 psii=acos(cos(inc)/cos(phii));

% Correct for sign ambiguity of the heading
   if pi/2<abs(node-theta) & abs(node-theta)<3*pi/2
      psii=-psii;

   end
```

```
% Calulate the value c from law of cosines (c is used for simplification
%   in the calculation of relative heading)
    c=(ve^2+vh^2-2*ve*vh*cos(psii))^.5;

% Calculate relative heading (psir)
   psir=acos((vh*cos(psii)-ve)/c);

% Correct for sign ambiguity of relative heading
   if pi/2<abs(node-theta) & abs(node-theta)<3*pi/2
     psir=-psir;
   end

% Calculate relative velocity (vr)
   vr=(vi^2+ve^2-2*vh*ve*cos(psii))^.5;

% The initial flight path angle relative to the rotating earth (gammar)
%   as a function of inertial velocity, inertial flight path angle and
%   relative velocity
   gammar=asin(vi*sin(gammai)/vr);

end

% Establish the initial condition vector
y0=[ri vr gammar thetai phii psir];

% Set initial and final time for integration.  Modify tf as needed to
%   include entire trajectory
 ti=0;
 tf=10000;

% Select set of differential equations based on shape of atmosphere
%   selected
if shape==1

% Options-y1s.m stops integration at surface of spherical Earth (sea
%   level), initial step size and maximum step size can be modified as
%   desired
   options=odeset('Events',@stops,'InitialStep',.0004,'MaxStep',1);
   [t Y]=ode45(@spherical,[ti tf],y0,[options],m,S,Cl,Cd,sigma);

else
% Options-y1.m stops integration at surface of ellipsoid Earth (sea level)
   options=odeset('Events',@stope,'InitialStep',.0004,'MaxStep',1);
   [t Y]=ode45(@ellipsoid,[ti tf],y0,[options],m,S,Cl,Cd,sigma);

end
```

```
% Calculate acceleration of vehicle (A) from equation of motion (dV)
  for i=1:length(t)
     A(i,1)=-((rhos*exp(-beta*(Y(i,1)-re))*S*Cd*Y(i,2)^2)/2)/m-g0*...
        (re/Y(i,1))^2*sin(Y(i,3))+Y(i,1)*omegae^2*cos(Y(i,3))*...
        (cos(Y(i,5))*sin(Y(i,3))-sin(Y(i,5))*sin(Y(i,6))*cos(Y(i,3)));
  end

if shape==1
% Plot Altitude/Time
   figure(1);plot(t,(Y(:,1)-re)/1000);
   xlabel('Time (sec)');
   ylabel('Altitude (km)');

% Plot deceleration/Altitude
   figure(2);plot((Y(:,1)-re)/1000,-A(:,1)/g0);
   xlabel('Altitude (km)');
   ylabel('Deceleration/g0');

% Plot flight-path angle/altitude
   figure(3);plot((Y(:,1)-re)/1000,Y(:,3)*180/pi);
   xlabel('Altitude (km)');
   ylabel('Relative Flight-Path Angle (deg)');

% Plot Longitude/altitude
   figure(4);plot((Y(:,1)-re)/1000,Y(:,4)*180/pi);
   ylabel('Longitude (deg)');
   xlabel('Altitude (km)');

% Plot Latitude/altitude
   figure(5);plot((Y(:,1)-re)/1000,Y(:,5)*180/pi);
   ylabel('North Latitude (deg)');
   xlabel('Altitude (km)');

% Plot relative velocity/altitude
   figure(6);plot((Y(:,1)-re)/1000,Y(:,2)/1000);
   xlabel('Altitude (km)');
   ylabel('Relative Velocity (km/sec)');

% Plot relative heading/altitude
   figure(7);plot((Y(:,1)-re)/1000,Y(:,6)*180/pi);
   ylabel('Heading Angle (deg)');
   xlabel('Altitude (km)');
else

% Plot Altitude/Time
```

```
    figure(1);plot(t,(Y(:,1)-rp/(cos(Y(length(t),5))*...
       ((tan(Y(length(t),5)))^2+(1-ee)^2)^.5))/1000);
    xlabel('Time (sec)');
    ylabel('Altitude (km)');

% Plot deceleration/Altitude
    figure(2);plot((Y(:,1)-rp/(cos(Y(length(t),5))*...
       ((tan(Y(length(t),5)))^2+(1-ee)^2)^.5))/1000,-A(:,1)/g0);
    xlabel('Altitude (km)');
    ylabel('Deceleration/g0');

% Plot flight-path angle/altitude
    figure(3);plot((Y(:,1)-rp/(cos(Y(length(t),5))*...
       ((tan(Y(length(t),5)))^2+(1-ee)^2)^.5))/1000,Y(:,3)*180/pi);
    xlabel('Altitude (km)');
    ylabel('Relative Flight-Path Angle (deg)');

% Plot Longitude/altitude
    figure(4);plot((Y(:,1)-rp/(cos(Y(length(t),5))*...
       ((tan(Y(length(t),5)))^2+(1-ee)^2)^.5))/1000,Y(:,4)*180/pi);
    ylabel('Longitude (deg)');
    xlabel('Altitude (km)');

% Plot Latitude/altitude
    figure(5);plot((Y(:,1)-rp/(cos(Y(length(t),5))*...
       ((tan(Y(length(t),5)))^2+(1-ee)^2)^.5))/1000,Y(:,5)*180/pi);
    ylabel('North Latitude (deg)');
    xlabel('Altitude (km)');

% Plot relative velocity/altitude
    figure(6);plot((Y(:,1)-rp/(cos(Y(length(t),5))*...
       ((tan(Y(length(t),5)))^2+(1-ee)^2)^.5))/1000,Y(:,2)/1000);
    xlabel('Altitude (km)');
    ylabel('Relative Velocity (km/sec)');

% Plot relative heading/altitude
    figure(7);plot((Y(:,1)-rp/(cos(Y(length(t),5))*...
       ((tan(Y(length(t),5)))^2+(1-ee)^2)^.5))/1000,Y(:,6)*180/pi);
    ylabel('Heading Angle (deg)');
    xlabel('Altitude (km)');

end
```

GUI Files

The graphical user interface (GUI) files allow intuitive and simple operation of the MATLAB program. They are called from the core program as needed.

Coordinate System and Atmosphere

The first GUI used by the software is the *inputs.m* file. This file allows the user to select a coordinate system for the initial known parameters from a list including the vehicle pointing system, the ECI system and classic orbital elements. It also allows the user to specify a spherical atmosphere or an ellipsoidal atmosphere. The GUI display is shown in Figure (52) followed by the MATLAB code for *inputs.m*.

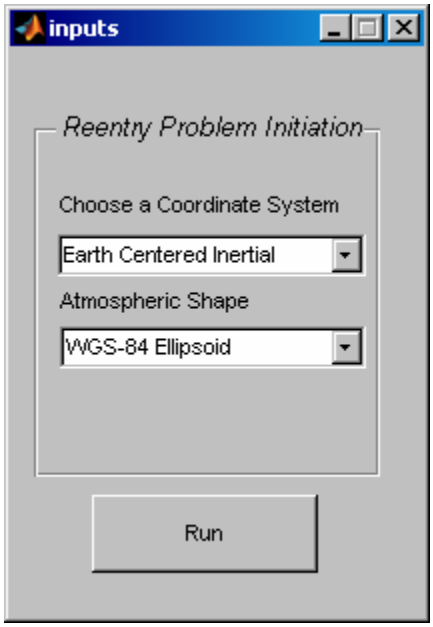

Figure 60: inputs.m GUI Display

```
function varargout = inputs(varargin)
% INPUTS M-file for inputs.fig
%      INPUTS, by itself, creates a new INPUTS or raises the existing
%      singleton*.
%
%      H = INPUTS returns the handle to a new INPUTS or the handle to
%      the existing singleton*.
%
```

```
%      INPUTS('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in INPUTS.M with the given input arguments.
%
%      INPUTS('Property','Value',...) creates a new INPUTS or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before inputs_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to inputs_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help inputs

% Last Modified by GUIDE v2.5 10-Jan-2006 15:31:22

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @inputs_OpeningFcn, ...
                   'gui_OutputFcn',  @inputs_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before inputs is made visible.
function inputs_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to inputs (see VARARGIN)
```

```
% Choose default command line output for inputs
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes inputs wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = inputs_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Convert user selection of coordinate system to value to be stored as
%   var.mat (vehicle pointing system=1, ECI system=2,
%   Classic orbital elements =3
str=get(hObject,'String');
val=get(hObject,'Value');
switch str{val};
   case'Vehicle Pointing'
      var=1;
   case'Earth Centered Inertial'
      var=2;
   case'Classic Orbital Elements'
      var=3;
end

save('var');

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1
```

```
% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Convert user selection of atmosphere shape to value to be stored as
%   shape.mat (spherical=1, ellipsoid=2)
str=get(hObject,'String');
val=get(hObject,'Value');
switch str{val};
   case'Spherical'
      shape=1;
   case'WGS-84 Ellipsoid'
      shape=2;
end

save('shape');

% Hints: contents = get(hObject,'String') returns popupmenu2 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu2


% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Close GUI when Run is selected
close('inputs')
```

Input Function for Vehicle Pointing System

The file *vpoint.m* sets up a graphical interface to allow the user to input the vehicle parameters for mass, reference surface area, coefficient of lift, and coefficient of drag. It also allows the user to specify the initial conditions for the equations of motion: radius, relative flight-path angle, relative velocity, geocentric latitude, longitude, relative heading, roll angle. The GUI display is shown in Figure (53) followed by the MATLAB code for *vpoint.m*.

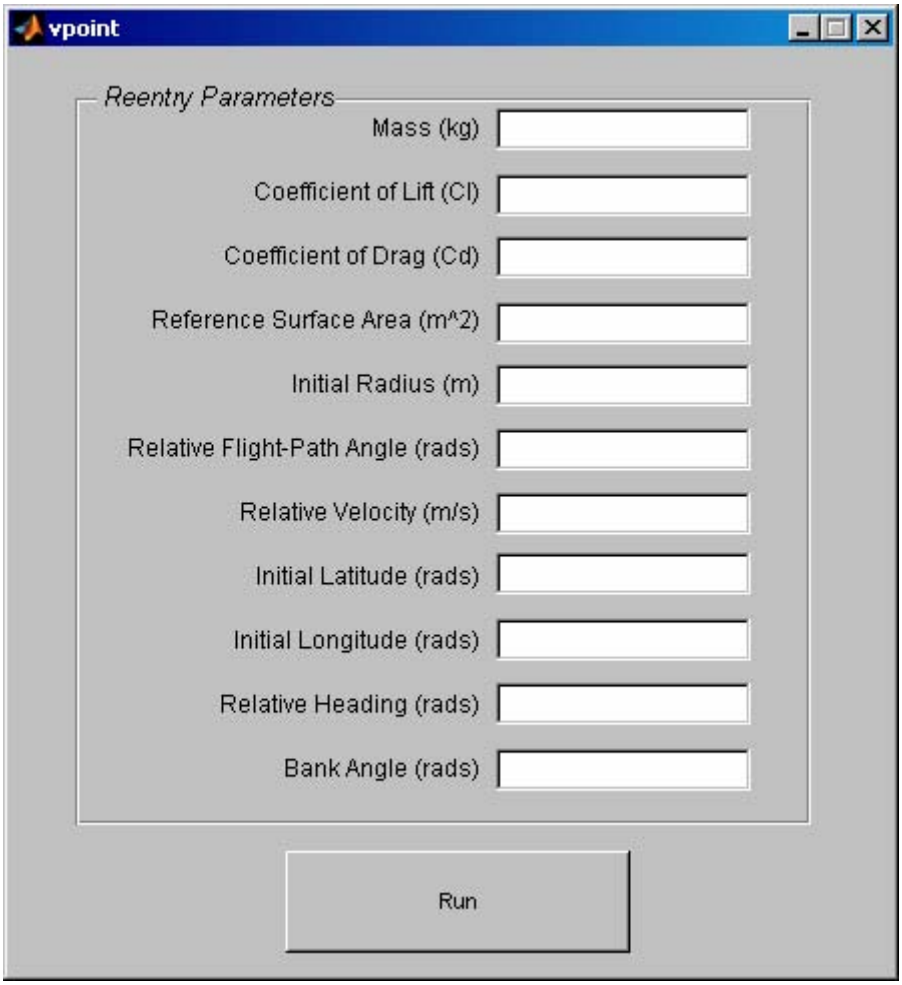

Figure 61. GUI to Input Parameters in Vehicle Pointing System

```
function varargout = vpoint(varargin)
% VPOINT M-file for vpoint.fig
%      VPOINT, by itself, creates a new VPOINT or raises the existing
%      singleton*.
%
%      H = VPOINT returns the handle to a new VPOINT or the handle to
%      the existing singleton*.
%
%      VPOINT('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in VPOINT.M with the given input arguments.
%
%      VPOINT('Property','Value',...) creates a new VPOINT or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before vpoint_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to vpoint_OpeningFcn via varargin.
```

```
%
%     *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%     instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help vpoint

% Last Modified by GUIDE v2.5 10-Jan-2006 16:37:44

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @vpoint_OpeningFcn, ...
                   'gui_OutputFcn',  @vpoint_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before vpoint is made visible.
function vpoint_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to vpoint (see VARARGIN)

% Choose default command line output for vpoint
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes vpoint wait for user response (see UIRESUME)
```

```
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = vpoint_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double

% Convert value for mass to matrix and save to m.mat
m=str2double(get(hObject,'String'));
save('m')

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit11_Callback(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text
%        str2double(get(hObject,'String')) returns contents of edit11 as a double

% Convert value for coefficient of lift to matrix and save to Cl.mat
```

```
Cl=str2double(get(hObject,'String'));
save('Cl');

% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit12_Callback(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit12 as text
%        str2double(get(hObject,'String')) returns contents of edit12 as a double

% Convert value for coefficient of drag to matrix and save to Cd.mat
Cd=str2double(get(hObject,'String'));
save('Cd');

% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit13_Callback(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit13 as text
%        str2double(get(hObject,'String')) returns contents of edit13 as a double

% Convert value for surface area to matrix and save to S.mat
S=str2double(get(hObject,'String'));
save('S');

% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit14_Callback(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit14 as text
%        str2double(get(hObject,'String')) returns contents of edit14 as a double

% Convert value for radius to matrix and save to ri.mat
ri=str2double(get(hObject,'String'));
save('ri');

% --- Executes during object creation, after setting all properties.
function edit14_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit15_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit15 as text
%        str2double(get(hObject,'String')) returns contents of edit15 as a double

% Convert value for relative flight-path angle to matrix and save to
%   gammar.mat
gammar=str2double(get(hObject,'String'));
save('gammar');

% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit16_Callback(hObject, eventdata, handles)
% hObject    handle to edit16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit16 as text
%        str2double(get(hObject,'String')) returns contents of edit16 as a double

% Convert value for relative velocity to matrix and save to vr.mat
vr=str2double(get(hObject,'String'));
save('vr');

% --- Executes during object creation, after setting all properties.
function edit16_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit17_Callback(hObject, eventdata, handles)
% hObject    handle to edit17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit17 as text
%        str2double(get(hObject,'String')) returns contents of edit17 as a double

% Convert value for geocentric latitude to matrix and save to phii.mat
phii=str2double(get(hObject,'String'));
save('phii');

% --- Executes during object creation, after setting all properties.
function edit17_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit18_Callback(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit18 as text
%        str2double(get(hObject,'String')) returns contents of edit18 as a double

% Convert value for longitude to matrix and save to thetai.mat
thetai=str2double(get(hObject,'String'));
save('thetai');

% --- Executes during object creation, after setting all properties.
function edit18_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit19_Callback(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit19 as text
%        str2double(get(hObject,'String')) returns contents of edit19 as a double

% Convert value for relative heading to matrix and save to psir.mat
psir=str2double(get(hObject,'String'));
save('psir');

% --- Executes during object creation, after setting all properties.
function edit19_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit20_Callback(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit20 as text
%        str2double(get(hObject,'String')) returns contents of edit20 as a double

% Convert value for bank angle to matrix and save to sigma.mat
sigma=str2double(get(hObject,'String'));
save('sigma');
```

```
% --- Executes during object creation, after setting all properties.
function edit20_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Close GUI when RUN pushbutton is pressed
close('vpoint');
```

Input Function for ECI System

The file *eci.m* sets up a graphical interface to allow the user to input the vehicle parameters for mass, reference surface area, coefficient of lift, and coefficient of drag. It also allows the user to specify the initial conditions in the ECI coordinate system: inertial position, inertial velocity, roll angle and Greenwich sidereal time. The GUI display is shown in Figure (54) followed by the MATLAB code for *eci.m*.

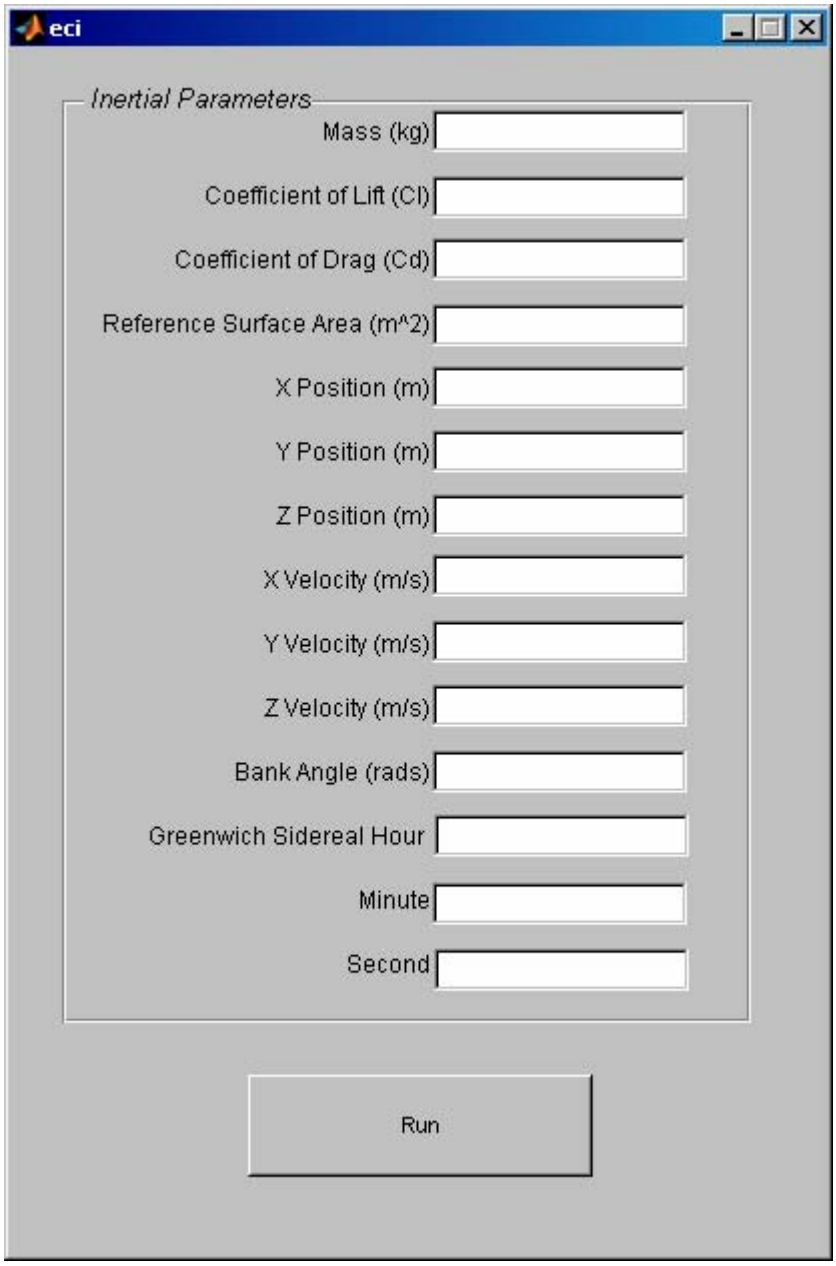

Figure 62. GUI to Input Parameters in ECI System

```
function varargout = eci(varargin)
% ECI M-file for eci.fig
%      ECI, by itself, creates a new ECI or raises the existing
%      singleton*.
%
%      H = ECI returns the handle to a new ECI or the handle to
%      the existing singleton*.
%
%      ECI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ECI.M with the given input arguments.
```

```
%
%      ECI('Property','Value',...) creates a new ECI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before eci_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to eci_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help eci

% Last Modified by GUIDE v2.5 10-Jan-2006 18:11:33

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @eci_OpeningFcn, ...
                   'gui_OutputFcn',  @eci_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before eci is made visible.
function eci_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to eci (see VARARGIN)

% Choose default command line output for eci
```

```
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes eci wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = eci_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double

% Convert value for mass to matrix and save as m.mat
m=str2double(get(hObject,'String'));
save('m')

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit11_Callback(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text
%        str2double(get(hObject,'String')) returns contents of edit11 as a double

% Convert value for coefficient of lift to matrix and save as Cl.mat
Cl=str2double(get(hObject,'String'));
save('Cl');

% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit12_Callback(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit12 as text
%        str2double(get(hObject,'String')) returns contents of edit12 as a double

% Convert value for coefficient of drag to matrix and save as Cd.mat
Cd=str2double(get(hObject,'String'));
save('Cd');

% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
```

```
function edit13_Callback(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit13 as text
%        str2double(get(hObject,'String')) returns contents of edit13 as a double

% Convert value for reference surface area to matrix and save as S.mat
S=str2double(get(hObject,'String'));
save('S');

% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit14_Callback(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit14 as text
%        str2double(get(hObject,'String')) returns contents of edit14 as a double

% Convert value for ECI x position to matrix and save as x.mat
x=str2double(get(hObject,'String'));
save('x');

% --- Executes during object creation, after setting all properties.
function edit14_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit15_Callback(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit15 as text
%        str2double(get(hObject,'String')) returns contents of edit15 as a double

% Convert value for ECI y position to matrix and save as y.mat
y=str2double(get(hObject,'String'));
save('y');

% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit16_Callback(hObject, eventdata, handles)
% hObject    handle to edit16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit16 as text
%        str2double(get(hObject,'String')) returns contents of edit16 as a double

% Convert value for ECI z position to matrix and save as z.mat
z=str2double(get(hObject,'String'));
save('z');

% --- Executes during object creation, after setting all properties.
function edit16_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit17_Callback(hObject, eventdata, handles)
% hObject    handle to edit17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit17 as text
%        str2double(get(hObject,'String')) returns contents of edit17 as a double

% Convert value for ECI x velocity to matrix and save as vx.mat
vx=str2double(get(hObject,'String'));
save('vx');

% --- Executes during object creation, after setting all properties.
function edit17_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit18_Callback(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit18 as text
%        str2double(get(hObject,'String')) returns contents of edit18 as a double

% Convert value for ECI y velocity to matrix and save as vy.mat
vy=str2double(get(hObject,'String'));
save('vy');
```

```
% --- Executes during object creation, after setting all properties.
function edit18_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit19_Callback(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit19 as text
%        str2double(get(hObject,'String')) returns contents of edit19 as a double

% Convert value for ECI z velocity to matrix and save as vz.mat
vz=str2double(get(hObject,'String'));
save('vz');

% --- Executes during object creation, after setting all properties.
function edit19_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit20_Callback(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit20 as text
%        str2double(get(hObject,'String')) returns contents of edit20 as a double
```

```
% Convert value for bank angle to matrix and save as sigma.mat
sigma=str2double(get(hObject,'String'));
save('sigma');

% --- Executes during object creation, after setting all properties.
function edit20_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit21_Callback(hObject, eventdata, handles)
% hObject    handle to edit21 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit21 as text
%        str2double(get(hObject,'String')) returns contents of edit21 as a double

% Convert value for Greenwich sidereal hour to matrix and save as hour.mat
hour=str2double(get(hObject,'String'));
save('hour');

% --- Executes during object creation, after setting all properties.
function edit21_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit21 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit22_Callback(hObject, eventdata, handles)
% hObject    handle to edit22 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit22 as text
%        str2double(get(hObject,'String')) returns contents of edit22 as a double

% Convert value for Greenwich sidereal minute to matrix and save as min.mat
min=str2double(get(hObject,'String'));
save('min');

% --- Executes during object creation, after setting all properties.
function edit22_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit22 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit23_Callback(hObject, eventdata, handles)
% hObject    handle to edit23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit23 as text
%        str2double(get(hObject,'String')) returns contents of edit23 as a double

% Convert value for Greenwich sidereal second to matrix and save as sec.mat
sec=str2double(get(hObject,'String'));
save('sec');

% --- Executes during object creation, after setting all properties.
function edit23_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Close GUI when RUN pushbutton is pressed
close('eci');
```

Input Function for Classic Orbital Elements

The file *orb.m* sets up a graphical interface to allow the user to input the vehicle parameters for mass, reference surface area, coefficient of lift, and coefficient of drag. It also allows the user to specify the initial conditions from the classic orbital elements: semi-major axis, true anomaly, inclination, eccentricity, argument of perigee, right ascension of the ascending node, roll angle and Greenwich sidereal time. The GUI display is shown in Figure (54) followed by the MATLAB code for *orb.m*.

Figure 63. GUI to Input Parameters from Classic Orbital Elements

```
function varargout = orb(varargin)
% ORB M-file for orb.fig
%      ORB, by itself, creates a new ORB or raises the existing
%      singleton*.
%
%      H = ORB returns the handle to a new ORB or the handle to
%      the existing singleton*.
%
```

```
%      ORB('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ORB.M with the given input arguments.
%
%      ORB('Property','Value',...) creates a new ORB or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before orb_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to orb_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help orb

% Last Modified by GUIDE v2.5 10-Jan-2006 18:07:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @orb_OpeningFcn, ...
                   'gui_OutputFcn',  @orb_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before orb is made visible.
function orb_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to orb (see VARARGIN)
```

```
% Choose default command line output for orb
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes orb wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = orb_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double

% Convert value for mass to matrix and save as m.mat
m=str2double(get(hObject,'String'));
save('m')

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit11_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text
%        str2double(get(hObject,'String')) returns contents of edit11 as a double

% Convert value for coefficient of lift to matrix and save as Cl.mat
Cl=str2double(get(hObject,'String'));
save('Cl');

% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit12_Callback(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit12 as text
%        str2double(get(hObject,'String')) returns contents of edit12 as a double

% Convert value for coefficient of drag to matrix and save as Cd.mat
Cd=str2double(get(hObject,'String'));
save('Cd');

% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

function edit13_Callback(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit13 as text
%        str2double(get(hObject,'String')) returns contents of edit13 as a double

% Convert value for reference surface area to matrix and save as S.mat
S=str2double(get(hObject,'String'));
save('S');

% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit15_Callback(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit15 as text
%        str2double(get(hObject,'String')) returns contents of edit15 as a double

% Convert value for semi-major axis to matrix and save as a.mat
a=str2double(get(hObject,'String'));
save('a');

% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit16_Callback(hObject, eventdata, handles)
% hObject    handle to edit16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit16 as text
%        str2double(get(hObject,'String')) returns contents of edit16 as a double

% Convert value for true anomaly to matrix and save as nu.mat
nu=str2double(get(hObject,'String'));
save('nu');

% --- Executes during object creation, after setting all properties.
function edit16_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit17_Callback(hObject, eventdata, handles)
% hObject    handle to edit17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit17 as text
%        str2double(get(hObject,'String')) returns contents of edit17 as a double

% Convert value for inclination to matrix and save as inc.mat
inc=str2double(get(hObject,'String'));
save('inc');

% --- Executes during object creation, after setting all properties.
function edit17_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit18_Callback(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit18 as text
%        str2double(get(hObject,'String')) returns contents of edit18 as a double

% Convert value for argument of perigee to matrix and save as omega.mat
omega=str2double(get(hObject,'String'));
save('omega');

% --- Executes during object creation, after setting all properties.
function edit18_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit19_Callback(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit19 as text
%        str2double(get(hObject,'String')) returns contents of edit19 as a double

% Convert value for right ascension of ascending node to matrix and save as
% node.mat
```

```
node=str2double(get(hObject,'String'));
save('node');

% --- Executes during object creation, after setting all properties.
function edit19_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit20_Callback(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit20 as text
%        str2double(get(hObject,'String')) returns contents of edit20 as a double

% Convert value for bank angle to matrix and save as sigma.mat
sigma=str2double(get(hObject,'String'));
save('sigma');

% --- Executes during object creation, after setting all properties.
function edit20_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit21_Callback(hObject, eventdata, handles)
% hObject    handle to edit21 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit21 as text
%        str2double(get(hObject,'String')) returns contents of edit21 as a double

% Convert value for Greenwich sidereal hour to matrix and save as hour.mat
hour=str2double(get(hObject,'String'));
save('hour');

% --- Executes during object creation, after setting all properties.
function edit21_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit21 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit22_Callback(hObject, eventdata, handles)
% hObject    handle to edit22 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit22 as text
%        str2double(get(hObject,'String')) returns contents of edit22 as a double

% Convert value for Greenwich sidereal minute to matrix and save as min.mat
min=str2double(get(hObject,'String'));
save('min');

% --- Executes during object creation, after setting all properties.
function edit22_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit22 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

function edit23_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit23 as text
%        str2double(get(hObject,'String')) returns contents of edit23 as a double

% Convert value for Greenwich sidereal second to matrix and save as sec.mat
sec=str2double(get(hObject,'String'));
save('sec');

% --- Executes during object creation, after setting all properties.
function edit23_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end


function edit24_Callback(hObject, eventdata, handles)
% hObject    handle to edit23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit23 as text
%        str2double(get(hObject,'String')) returns contents of edit23 as a double

% Convert value for eccentricity to matrix and save as e.mat
e=str2double(get(hObject,'String'));
save('e');

% --- Executes during object creation, after setting all properties.
function edit24_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Close GUI when RUN pushbutton is pressed
close('orb');
```

Ordinary Differential Equation Integrators

The core program *reentry.m* calls one of two functions to perform the integration of the equations of motion. The function *spherical.m* is called when the atmosphere selected by the user is spherical and the function *ellipsoid.m* is called when the user selects an ellipsoidal atmosphere. In both cases Equations (36) and (37) for calculating drag and lift are embedded in the equations of motion. In the function *ellipsoid.m* Equations (44) and (45), used to calculate the Earth's radius at a given geocentric latitude, are also embedded into the equations of motion.

Code for Spherical Atmosphere Integrator

```
% This integration function is called by reentry.m when a spherical atmosphere is
%       selected
function dy=reentry(t,y,m,S,Cl,Cd,sigma)

% Earth's acceleration from gravity at equatorial radius (m/s^2)
g0=9.798;

% Surface density of atmosphere at sea level (kg/m^3)
rhos=1.54793;

% Radius of Earth (m)
re=6378137;
```

```
% Inverse of the scale height of atmosphere (1/m)
beta=1/6935;

% WGS-84 rotation rate of Earth (rad/s)
omegae=.000072921151467;

% Establishing dy vector for equations of motion
dy=zeros(6,1);

% The equations of motion

% dr/dt (solved for radius)
dy(1)=y(2)*sin(y(3));

% dV/dt (solved for relative velocity)
dy(2)=-((rhos*exp(-beta*(y(1)-re))*S*Cd*y(2)^2)/2)/m-g0*(re/y(1))^2*...
   sin(y(3))+y(1)*omegae^2*cos(y(3))*(cos(y(5))*sin(y(3))-sin(y(5))...
   *sin(y(6))*cos(y(3)));

% dgamma/dt (solved for relative flight-path angle)
dy(3)=((rhos*exp(-beta*(y(1)-re))*S*Cl*y(2)^2)/2)*cos(sigma)/(y(2)*m)-...
   g0*(re/y(1))^2*cos(y(3))/y(2)+y(2)*cos(y(3))/y(1)+y(1)*omegae^2*...
   cos(y(5))*(cos(y(5))*cos(y(3))+sin(y(5))*sin(y(6))*sin(y(3)))/y(2)...
      +2*omegae*cos(y(5))*cos(y(6));

% dtheta/dt (soved for longitude)
dy(4)=y(2)*cos(y(3))*cos(y(6))/(y(1)*cos(y(5)));

% dphi/dt (solved for latitude)
dy(5)=y(2)*cos(y(3))*sin(y(6))/y(1);

% dpsi/dt (solved for relative heading)
dy(6)=((rhos*exp(-beta*(y(1)-re))*S*Cl*y(2)^2)/2)*sin(sigma)/(m*...
   cos(y(3))*y(2))-y(2)*cos(y(3))*cos(y(6))*tan(y(5))/y(1)+2*omegae*...
   (sin(y(6))*cos(y(5))*tan(y(3))-sin(y(5)))-y(1)*omegae^2*sin(y(5))*...
   cos(y(5))*cos(y(6))/(cos(y(3))*y(2));
```

Code for Ellipsoidal Atmosphere Integrator

```
% This integration function is called by reentry.m when an ellipsoidal
%   atmosphere is selected
function dy=reentry(t,y,m,S,Cl,Cd,sigma)

% Earth's acceleration due to gravity at equatorial radius (m/s^2)
```

```
g0=9.798;

% Earth's atmosphere's surface density (kg/m^3)
rhos=1.54793;

% WGS-84 value for the radius of the Earth at the equator (m)
re=6378137;

% WGS-84 value for the radius of the Earth at the poles (m)
rp=6356752.3142;

% Ellipticity of the Earth (e)
e=1-rp/re;

% Inverse of the scale height for Earth's atmosphere (1/m)
beta=1/6935;

% WGS-84 value for Earth's rotation rate (rad/s)
omegae=.000072921151467;

% Establishing dy vector for equations of motion
dy=zeros(6,1);

% The equations of motion

% dr/dt (solved for radius)
dy(1)=y(2)*sin(y(3));

% dV/dt (solved for relative velocity)
dy(2)=-((rhos*exp(-beta*(y(1)-rp/(cos(y(5))*((tan(y(5)))^2+...
   (1-e)^2)^.5)))*S*Cd*y(2)^2)/2)/m-g0*(re/y(1))^2*sin(y(3))+y(1)*...
   omegae^2*cos(y(3))*(cos(y(5))*sin(y(3))-sin(y(5))*sin(y(6))*cos(y(3)));

% dgamma/dt (solved for relative flight-path angle)
dy(3)=((rhos*exp(-beta*(y(1)-rp/(cos(y(5))*((tan(y(5)))^2+...
   (1-e)^2)^.5)))*S*Cl*y(2)^2)/2)*cos(sigma)/(y(2)*m)-g0*(re/y(1))^2*...
   cos(y(3))/y(2)+y(2)*cos(y(3))/y(1)+y(1)*omegae^2*cos(y(5))*...
   (cos(y(5))*cos(y(3))+sin(y(5))*sin(y(6))*sin(y(3)))/y(2)+2*omegae*...
   cos(y(5))*cos(y(6));

% dtheta/dt (solved for longitude)
dy(4)=y(2)*cos(y(3))*cos(y(6))/(y(1)*cos(y(5)));

% dphi/dt (solved for geocentric latitude)
dy(5)=y(2)*cos(y(3))*sin(y(6))/y(1);
```

```
% dpsi/dt (solved for relative heading)
dy(6)=((rhos*exp(-beta*(y(1)-rp/(cos(y(5))*((tan(y(5)))^2+...
   (1-e)^2)^.5)))*S*Cl*y(2)^2)/2)*sin(sigma)/(m*cos(y(3))*y(2))-y(2)*...
   cos(y(3))*cos(y(6))*tan(y(5))/y(1)+2*omegae*(sin(y(6))*cos(y(5))*...
   tan(y(3))-sin(y(5)))-y(1)*omegae^2*sin(y(5))*cos(y(5))*cos(y(6))/...
   (cos(y(3))*y(2));
```

Functions to Terminate Integration

The final two functions in the MATLAB software package are used to halt the integration functions when the entry vehicle reaches the surface of the Earth. The function *stops.m* is called by the core program *reentry.m* when a spherical atmosphere is selected by the user. The function is established by the *odeset* command as an "Event" and stops the integration function *spherical.m* when the radius solved for in the first equation of motion is less than the constant radius of the Earth. Likewise, the "Event" *stope.m* is implemented when an ellipsoidal atmosphere is selected. *stope.m* halts integration when the radius solved for in the first equation of motion is less than the radius of the Earth found in Equations (44) and (45) as a function of geocentric latitude.

Termination for Spherical Earth

```
%stops.m
%Events function to halt integration at radius of spherical Earth

function [value,isterminal,direction] = stops(t,y,m,S,Cl,Cd,sigma)

% Detect point that radius is less than radius of Earth
if y(1) < 6378137;
   value = 0;
else
   value = 1;
end

% Halt integration when value=0 is detected
isterminal = 1;
```

```
% Halt for all value=0
direction = 0;
```

Termination for Ellipsoidal Earth

```
%stope.m
%Events function to halt integration at radius of ellipsoidal Earth

function [value,isterminal,direction] = stope(t,y,m,S,Cl,Cd,sigma)

% Detect point that radius is less than radius of Earth
if y(1) < 6356752.3142/(cos(y(5))*((tan(y(5)))^2+...
       (6356752.3142/6378137)^2)^.5);
   value = 0;
else
   value = 1;
end

% Halt integration when value=0 is detected
isterminal = 1;

% Halt for all value=0
direction = 0;
```

Bibliography

1. Albrecht, Meredith M. *The Effect of Aerodynamic Surfaces Versus Thrust Maneuvers on Reentry Vehicles*. Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 2005 (GAE/ENY/05-S01).

2. Brauer, G.L., D.E. Cornick and R. Stevenson. *Capabilities and Applications of the Program to Optimize Simulated Trajectories (POST)*. CR-2770. Washington, D. C.: National Aeronautics and Space Administration, 1977.

3. European Organization for the Safety of Air Navigation and Institute of Geodesy and Navigation. "WGS 84 Implementation Manual." Report to International Civil Aviation Organization (ICAO). February 1998.

4. "Department of Defense World Geodetic System 1984." TR-8350.2. Report to National Imagery and Mapping Agency (NIMA), Bethesda MD. January 2000.

5. Duncan, Robert C. *Dynamics of Atmospheric Entry*. New York: McGraw-Hill Book Company Inc, 1962.

6. Griffin, Michael D., and French, James R. *Space Vehicle Design*. Reston VA: American Institute of Aeronautics and Astronautics, Inc., 2004.

7. Hicks, Kerry D. "Simple Modeling Tools for Force Applications Reentry Vehicles." Report to Phillips Laboratory Plans and Programs Directorate, Edwards AFB, CA. June 1993.

8. Hicks, Kerry D. *Introduction to Astrodynamic Reentry*. Unpublished text for MECH 637, Astrodynamic Reentry. Air Force Institute of Technology, Wright-Patterson AFB OH, 3 January 2006.

9. Indiana University Stat/Math Center. "A General Overview of MATLAB." n.pag. http://www.indiana.edu/~statmath/math/matlab/overview.html. 10 June 2005.

10. Longuski, James Michael, and Vinh, Nguyen Xuan. *Analytic Theory of Orbit Contraction and Ballistic Entry into Planetary Atmospheres*. Pasadena CA: The Jet Propulsion Laboratory, 1 September 1980 (JPL Publication 80-58).

11. The MathWorks. "Geocentric to Geodetic Latitude." n.pag. http://www.mathworks.com/access/helpdesk/help/toolbox/aeroblks/geocentrictogeodeticlatitude.html. 31 December 2005.

12. Murray, James E., and Tartabini, Paul V. *Development of a Mars Airplane Entry, Descent, and Flight Trajectory*. TM-2001-209035. National Aeronautics and Space Administration, 2001.

13. National Air and Space Intelligence Agency (NASIC). "IMPULSE Engineering Manual." Unpublished document accompanying IMPULSE software. Wright-Patterson Air Force Base OH, 2005.

14. Regan, Frank J. *Re-Entry Vehicle Dynamics*. AIAA Education Series: New York, 1984.

15. Regan, Frank J., and Anandakrishnan, Satya M. *Dynamics of Atmospheric Re-Entry*. AIAA Education Series: Washington, DC, 1993.

16. Vinh, Nguyen X., and others. *Hypersonic and Planetary Entry Mechanics.* Ann Arbor: The University of Michigan Press, 1980.

17. Way, David W., et al. *Aerocapture Simulation and Performance for the Titan Explorer Mission.* 2003-4951. American Institute of Aeronautics and Astronautics, 2003.

18. Weisel, William E. *Spaceflight Dynamics*. Boston: The McGraw-Hill Companies, Inc., 1997.

Vita

Captain Robert Earl Jameson Jr. graduated from Sahuaro High School in Tucson, Arizona in 1987. He enlisted in the Air Force in 1990 and was trained as a Broadcast Communications Technician at Lowry AFB, Colorado. He then served with Combat Camera at Norton AFB and March AFB in California, including deployments to Somalia and Haiti. His next assignment was at the Air Force News Agency at Kelly AFB, Texas. During this assignment in 1999 he graduated from Southwest Texas State University with a Bachelor of Applied Arts and Science.

In 2000 he graduated from Officer Training School and was commissioned as a Second Leiutenant. After training at Vandenberg AFB, he spent four years as a missile combat crew member and flight commander at F. E. Warren AFB, Wyoming. In September, 2005 he entered the Air Force Institute of Technology as a graduate student in the Space Systems Engineering program. Upon graduation in March, 2006 he will be assigned to the 2nd Range Operations Squadron at Vandenberg AFB.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
23-03-2006	**Master's Thesis**	Sep 2004 – Mar 2006

4. TITLE AND SUBTITLE
Development and Validation of Reentry Simulatin Using MATLAB

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Jameson, Robert E. Jr., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/ENY)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GSS/ENY/06-M08

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
NA

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research effort develops a program using MATLAB to solve the equations of motion for atmospheric reentry and analyzes the validity of the program for use as a tool to expeditiously predict reentry profiles. The reentry vehicle is modeled as a point mass with constant aerodynamic properties as defined by the user. The equations of motion for reentry are based on the two-body problem. The atmosphere is modeled as a single layer exponentially decreasing in density. The MATLAB program has the ability to derive the initial trajectory conditions from the position and velocity relative to the rotating surface of the Earth, the Earth-centered inertial position and velocity, or the classic orbital elements. The program is compared to previously established programs in order to validate its accuracy and numerical stability when predicting various reentry profiles to include sub-orbital, super-circular and hyperbolic trajectories as well as wide ranges of aerodynamic properties
.

15. SUBJECT TERMS
Reentry Vehicle, MATLAB, Trajectory Simulation, Equations of Motion, Ballistic Reentry, Gliding Reentry

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT					Lt Col Kerry D. Hicks
U	**U**	**U**	**UU**	146	**19b. TELEPHONE NUMBER** *(Include area code)* 937-255-3636 x4568, email:kerry.hicks@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18